Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Mitja Gorišek

Krdelo volkov: modeliranje in simulacija organiziranega plenjenja

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Iztok Lebar Bajec

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi preučite vir R. Escobedo et al (2014), Group size, individual role differentiation and effectiveness of cooperation in a homogeneous group of hunters, DOI: 10.1098/rsif.2014.0204 in poustvarite predstavljeni algoritem krdela volkov in plena. V nadaljevanju najprej ponovite eksperimente, nato pa poizkusite delovanje algoritma nadgraditi in izboljšati. Rezultate kritično komentirajte.

Zahvaljujem se svoji družini, Kaji za vso pomoč in podporo med študijem. Zahvala gre tudi mentorju izr. prof. dr. Iztoku Lebarju Bajcu za vso pomoč pri izdelavi diplomske naloge.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Uporabljena orodja in tehnologije **3**

2.1 Unity 3

2.2 MonoDeveloped in Visual Studio 2015 4

2.3 Photoshop 4

2.4 MatLab 5

2.5 Autodask Maya 2013 5

2.6 Trigonometrija 6

3 Simulacija plenjenja **7**

3.1 Opis modela v simulaciji 7

3.2 Predstavitev modela simulacije 8

4 Izdelava simulacije **15**

4.1 Ustvarjanje projekta 15

4.2 Ustvarjanje scene 18

4.3 Prvi del diplomske naloge 18

4.4 Drugi del diplomske naloge 21

4.5 Urejevalnik postavitev 25

4.6 Vizualizacija 27

5 Rezultati in testi **31**
5.1 Rešitev napačne formule 31
5.2 Problem poravnave 34
5.3 Primerjava napada na plen z in brez alfa para 35
5.4 Zanimivosti 41

6 Sklepne ugotovitve **43**

Literatura **45**

Povzetek

Naslov: Krdelo volkov: modeliranje in simulacija organiziranega plenjenja

Avtor: Mitja Gorišek

Lov in gibanje živali v večjem številu je zelo zanimiv fenomen, ki ga opazujemo že tisočletja. Preučevanje tehnik plenjenja in gibanja sega v dobo pračloveka, kjer je sam človek poustvarjal tehnike, ki jih je osvojil z opazovanjem živali pri lovu. Da bi povečal svojo uspešnost je imitiral videno in poskušal izboljšati rezultate. V diplomski nalogi poskušamo na takšen način ustvariti model plenjenja živali, sestavljen iz preprostih pravil, kot ga uporabljajo volkovi. Za izdelavo simulacije se zgledujemo po viru R. Escobedo et al (2014), Group size, individual role differentiation and effectiveness of cooperation in a homogeneous group of hunters, DOI: 10.1098/rsif.2014.0204, na podlagi katerega izdelamo algoritem in se poskušamo približati rezultatom. Za priložen algoritem implementiramo nadgradnjo in izboljšave. Tehnike plenjenja se danes uporabljajo v različnih modelih, simulacijah, filmski in igralni industriji.

Ključne besede: simulacija plenjenja živali, plenjenje, obkoljevanje.

Abstract

Title: Wolf pack: modelling and simulation of organized hunting

Author: Mitja Gorišek

Hunting and animal movement in bigger numbers is very interesting phenomenon, which we are spectating for thousands of years. Studying of hunting and movement technique is reaching back in age of primitive man. Human recreated observed hunting techniques. To increase his hunting success, he imitated observed behaviors and tried to improve it. In the thesis we try to use this approach to create hunting model, which is made with simple rules, similar to wolfs. The simulation is recreated from article R. Escobedo et al (2014), Group size, individual role differentiation and effectiveness of cooperation in a homogeneous group of hunters, DOI: 10.1098/rsif.2014.0204. In thesis we try to compare results, upgrade and improve the algorithm. This techniques are used in various models, simulations, movie and gaming industry.

Keywords: simulation of hunting animals, hunt, encircle.

Poglavje 1

Uvod

Preučevanje lova in gibanje živali sega tisočletja nazaj vse do pračloveka, ki je z opazovanjem živali skušal razumeti tehnike plenjenja. Človek je hitro postal lovec, z opazovanjem plenilcev pa je kmalu spoznal, da lahko živali ujame in jih udomači, ter s tem razvije nove panoge.

Volkovi zelo redko živijo samotarsko življenje, saj so zelo teritorialni. Območje plenjenja le redko zadostuje dovolj majhnih živali za tako velikega plenilca. Skupino volkov imenujemo krdel, kjer povprečna skupina šteje od 5 do 11 članov. Živali so kot individualne izpostavljene nevarnosti, zato je njihov naravni nagon, da se zbirajo v skupinah kadar se počutijo ogrožene in si s tem povečajo možnost preživetja pred plenilci. Tekmovanje za čim boljši položaj v čredi, jati ali neki skupini je neprestano, saj vsak osebek skrbi zase. Takšno obnašanje skupine kot celote zmede plenilce, ker je sledenje posameznemu osebku zelo težko. Skupinsko plenjenje je pogosta oblika sodelovanja živali, ki se pojavijo v različnih stopnjah zahtevnosti. Sodelovanje večjih plenilcev skupaj, poveča možnost ulova plena, ki ga kot posameznik ne bi nikoli uspešno ulovil.

V skupini živali plenilci poskušajo odkriti šibki člen, ki ponavadi zaostaja, je utrujen ali ranjen. Primerno tarčo pričnejo preganjati, ločijo jo od črede in obkolijo. Postavijo se v krogu oddaljeni na minimalni varni razdalji, nato jo utrudijo. V naslednjih poglavjih diplomskega dela je predstavljen poe-

nostavljen algoritem plenjenja, ki je sestavljen z nekaj preprostimi pravili. Obnašanje, ki ga poskušamo prikazati je zgolj približek, saj tematika plenjenja ni dovolj znanstveno raziskana, da bi lahko točno definirali obnašanje plenilcev, zato se lahko naravnemu fenomenu zgolj približamo. Algoritem v simulaciji vsebuje en primerek plena, ki predstavlja šibek člen, že odmaknjen od ostalih, ter skupino plenilcev, ki ga poskušajo uloviti.

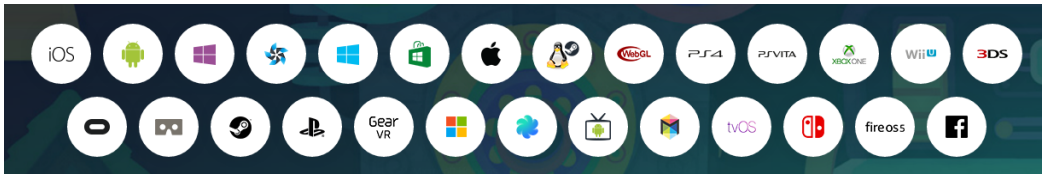
V prvem delu diplomske naloge predstavimo in poustvarimo algoritem obkoljevanja plena po viru [1]. V drugem delu diplomske naloge algoritem izboljšamo in nadgradimo z metodo napada na plen, ter na podlagi prvega dela diplomske naloge ustvarimo hierarhijo v skupini plenilcev. S testi poskušamo primerjati uspešnost skupine plenilcev brez in z alfa parom, kjer alfa par definiramo kot plenilca alfa1 in alfa2, ki imata izboljšane lastnosti. V zadnjem delu pa zapišemo ugotovitve rezultatov analiziranih testov.

Poglavje 2

Uporabljena orodja in tehnologije

2.1 Unity

Je programska oprema, namenjena za razvijanje iger, spletnih aplikacij in lažjemu razvoju simulacij. Gre za igralni pogon (angl. *game engine*), ki ga je razvilo podjetje Unity Technologies. Prednost te programske opreme je prenosljivost projekta med več kot 20 različnimi platformami (glej sliko 2.1). Unity v letu 2017 ponuja štiri različne artikle in sicer Personal, Plus, Pro in Enterprise [2]. Različica personal je namenjena posameznikom in manjšim podjetjem, ki zaslužijo letno manj kot 100.000$. V diplomsko nalogi uporabljamo verzijo 5.5.0. Na voljo imamo 3 programske jezike JavaScript, Boo in C#.



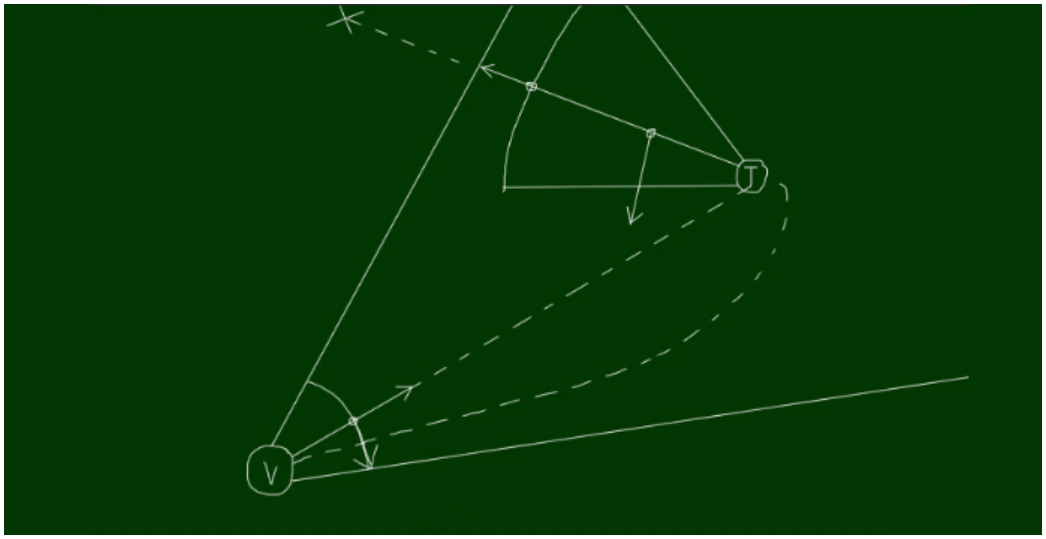
Slika 2.1: Nekaj različnih platform, ki jih podpira Unity.

2.2 MonoDeveloped in Visual Studio 2015

Oba programa sta enostavna urejevalnika za pisanje programske kode, ki nam omogočata potrebne funkcije za pisanje C# kode v Unity. Ponujata poganjanje projekta znotraj programa, razhroščevanje (angl. *debug*), samo dokončanje kode (angl. *autocomplete*), itd. Ob namestitvi Unity je zraven priložen MonoDeveloped, ki je zelo preprost urejevalnik za pisanje skriptnih jezikov. Urejevalnik je zelo pregleden, ima funkcijo za samo dokončanje kode, kar poveča produktivnost. Programska oprema ima majhno pomanjkljivost, saj je razhroščevanje zelo nepregledno zato je za nadalnje kodiranje uporabljen Visual Studio 2015. Z njihove uradne spletne strani snamemo zastonjsko različico Visual Studio 2015. Za lažje delo namestimo tudi Visual Studio 2015 Tools for Unity, ki ga snamemo iz Visual Studio spletne trgovine [3]. To je zastonjski dodatek, ki nam omogoči poganjanje Unity projekta znotraj programa Visual Studio, omogoči nam enostavno razhroščevanje, medtem ko poganjamo projekt.

2.3 Photoshop

Photoshop je program namenjen manipulaciji nad slikami, izdelavi animacij in risanju [4]. Program je zelo močno orodje, zato je zelo priljubljen med grafičnimi oblikovalci, umetniki, filmski in igralni industriji, saj omogoča enostavno izdelavo tekstur za 2D ali 3D grafične modele. Uporabili smo ga za prototipiranje, vizualizacijo ideje, obdelavo slik in izdelavo grafov iz simulacije (glej sliko 2.2).



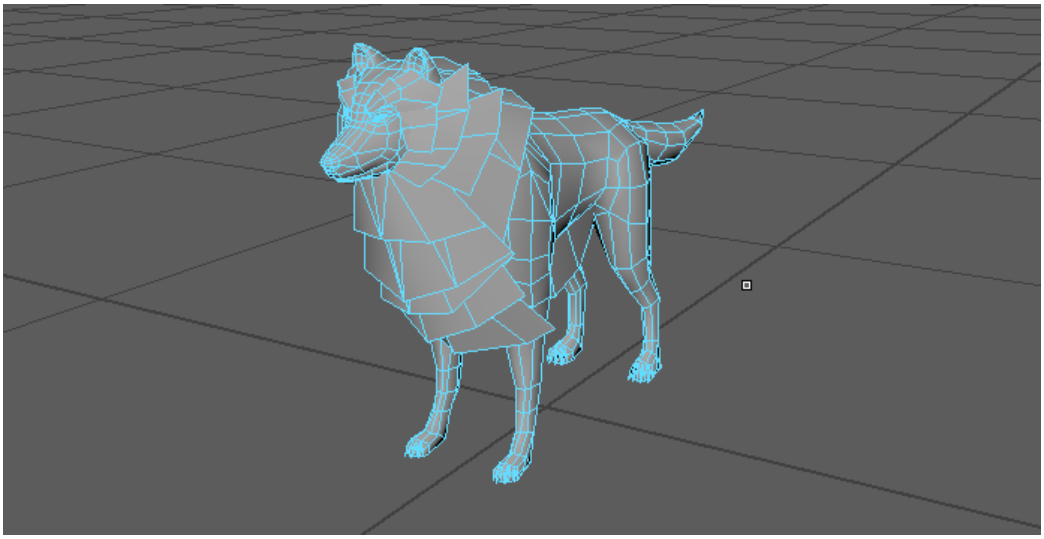
Slika 2.2: Prototip ideje.

2.4 MatLab

Je program za reševanje matematičnih in znanstvenih problemov [5]. Program smo uporabili za izdelavo grafov na podlagi pridobljenih rezultatov iz simulacije. Uporabili smo demo verzijo R2016b (9.1.0.441655).

2.5 Autodask Maya 2013

Je izvrsten program za izdelavo 3D modelov, animiranje in izdelavo vizualnih efektov za različne simulacije, igre, filme in mnoge druge namene. Najpogosteje se uporablja v filmski in igralni industriji za izdelavo 3D modelov, na katere kasneje prilepimo teksture, ki so izdelane v grafičnem programu, kot je na primer Photoshop. Maya nam omogoča pisanje skript v C# in mnogih drugih jezikih, s katerimi lahko ustvarimo vizualne efekte [6]. S programom so bili preurejeni modeli za plen in plenilca, ki jih uvozimo v Unity oblike *.obj* in jih kasneje uporabimo v simulaciji (glej sliko 2.3).



Slika 2.3: 3D model plenilca v podobi volka.

2.6 Trigonometrija

Simulacija je izvedena v 3D svetu, zato je vektor predstavljen kot zaporedje treh števil X, Y in Z, v zapisu enojne natančnosti (angl. *float*). Plen, plenilce in vizualne efekte v simulaciji predstavimo kot vektor na koordinatnem sistemu v obliki točk. Simulacija deluje na podlagi sil, zato bomo za premikanje objektov uporabljali osnovne vektorske operacije: seštevanja, odštevanja, množenje vektorja s skalarjem, normaliziranje, itd. Unity ponuja dobro dokumentacijo zato osnovnih operacij za računanje z vektorji ne poustvarjamo.

Seštevanje ali odštevanje dveh vektorjev nam vrne vektor, s katerim lahko simuliramo premik objekta v simulaciji bodisi plena, plenilca ali pomožnih objektov za vizualni prikaz. Pove nam v katero smer se premaknemo in kako daleč. Prav tako nam nudi osnovo za delo s silami. Z evklidsko razdaljo lahko izračunamo razdaljo med dvema vektorjema. V simulaciji jo zelo pogosto uporabimo za preverjanje razdalje med objekti v sceni. Normaliziranje vektorja nam nudi lažje računanje z vektorji. Hitrost premikanja lahko omejimo z normalizacijo in množenjem vektorja s skalarjem.

Poglavje 3

Simulacija plenjenja

Plenilstvo je poseben odnos živali, kjer en organizem (plenilec) ubije drugega (plen) z različnih nagonskih razlogov. Plenjenje živali ne koristi samo plenilcem, ampak tudi ekosistemu, saj s tem uravnavajo ravnovesje populacije ostalih živali. V simulaciji volkovi predstavljajo plenilce, plen pa različne živali. V tem poglavju bomo predstavili podrobnosti delovanja modela skupinskega vedenja plenilcev, ki smo ga implementirali na podlagi vira [1] .

3.1 Opis modela v simulaciji

Simulacija je sestavljena iz več različnih funkcij, ki skupaj sestavljajo algoritem. Razdeljen je na dva dela, prvi del je obkoljevanje, ki ga želimo poustvariti po priloženem viru [1], drugi del pa predstavlja plenjenje, ki ga načrtujemo sami.

Pri simulaciji obkoljevanja poskušamo poustvariti način gibanja plenilcev in plena. Ugotoviti želimo kaj vse vpliva na plenilce in kako se odločajo, ter na kakšen način obkolijo plen. Hkrati nas zanima kakšni dejavniki vplivajo na plen in odločanje le tega glede na okoliščine. Vir [1] opisuje lov plenilcev na posamezen plen, ki predpostavlja, da so plenilci plen že ločili od skupine, nato se želijo plenu približati in ga obkoliti.

Kadar govorimo o takšnih modelih, se velikokrat osredotočimo na simu-

lacije s silami, kjer na osebke vpliva sila kohezije (angl. *cohesion*), razmika (angl. *separation*) in poravnave (angl. *alignment*), ki jih je prvi predstavil Reynolds [7]. Takšen sistem lahko zadosti velikim simulacijam, kot je simuliranje jate ptic in rib, poenostavljeno čredenje z ovčarjem in ovcami, itd. V našem primeru pa to ne pride v poštev, saj se model kot takšen ne bo dovolj dobro približal poustvarjanju simulacije obkoljevanja glede na priložena pravila iz vira [1].

V drugem delu simulacije izvedemo napad na plen. Uprizoriti želimo fazo, kjer so plenilci že obkolili plen in v tej točki pričnejo z napadanjem nanj. Ugotoviti želimo kakšni dejavniki vplivajo na dogodek in ali ima skupina z alfa parom kakšno prednost pri lovu pred skupino, ki je enakovredna. V tem delu poskušamo simulirati napad plenilca na plen in branjenje plena pred plenilci.

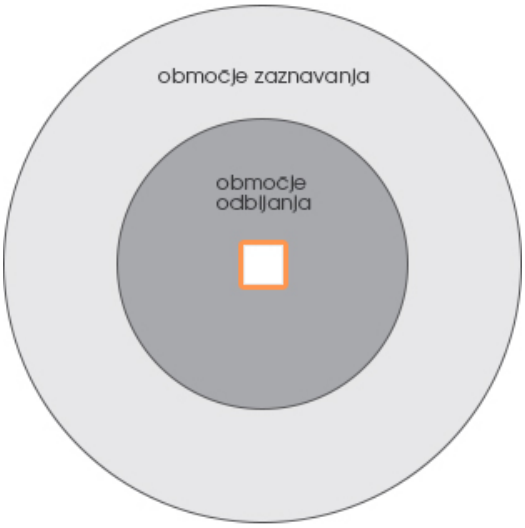
Iz prejšnjega poglavja lahko razberemo, da algoritem simulacije temelji na podlagi sil. Sile zapišemo v obliki usmerjenega vektorja in jih prištejemo k trenutni lokaciji objektov, ter s tem ustvarimo gibanje. Plenjenje je zelo kompleksen fenomen, zato so pravila algoritma poenostavljena. Algoritem plenjenja bomo poskušali upodobiti z nekaj preprostimi pravili, rezultat pa prikažemo kot simulacijo v igralnem pogonu Unity. Simulacija vsebuje več različnih testnih primerov pregona živali, zato se plenilci in plen v posameznih primerih različno obnašajo. Glavni cilj prvega dela simulacije je, da plenilci ujamejo plen, v drugem delu pa ga želimo v čim krajšem času onesposobiti.

3.2 Predstavitev modela simulacije

Model smo razvili v 3 dimenzionalnem svetu, kjer je vsak posameznik opisan z lokacijo $\mathbf{u}_i$ in smernim vektorjem gibanja $\mathbf{v}_i$, ki je zapisan s koordinatami X, Y in Z. Simulacija teče s časovnim korakom 0.02s, ki v vsakem koraku pridobi informacije iz okolja, na podlagi katerih izračunamo vpliv na vektor gibanja za posamezni osebek.

Simuliranje temelji na območju zaznavanja osebkov [7]. Vsak osebek ima

določena območja, na podlagi katerih se odloča kaj bo storil (območje zaznavanja plenilca, območje kohezije, poravnave, razmika, itd). V našem primeru pa območja delimo le na območje zaznavanja *vd*, kjer zaznamo plenilca in območje odbijanja *cd*, kjer se zgodijo akcije napada, obrambe ter odbijanja (glej sliko 3.1).



Slika 3.1: Območji zaznavanja.

3.2.1 Opis plenilca

V prvem delu simulacije je osnovna naloga plenilca, ujeti plen. Iz priloženega vira [1] razberemo pravila obnašanja plenilca:

Prvo pravilo:

- Približuj se plenu, dokler ne dosežemo minimalne varnostne razdalje.
- Vsak plenilec se približuje plenu v ravni liniji, neodvisno od ostalih plenilcev.
- Vsak plenilec preneha s približevanjem na razdalji *dc*.

Drugo pravilo:

- Ko dosežemo minimalno varnostno razdaljo *dc*, se plenilci pomaknejo stran od ostalih plenilcev, ki so prav tako na varnostni razdalji in tako simuliramo obkoljevanje.

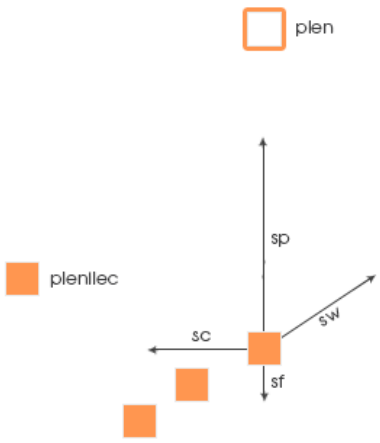
Interakcija med plenilcem in plenom je asimetrična, saj plenilci vedno odbijajo plen, plen pa plenilce vedno privlači. Ko plenilci prispejo na varno razdaljo *dc*, prenehajo s premikanjem proti plenu. Bližje kot je plenilec plenu, večja je odbojna sila in dlje kot je od plena manjša je privlačnost.

V našem modelu je interakcija med plenilci zgolj odbojne narave. Plenilci se na razdalji *da*, pričnejo odmikati drug od drugega. Odbojnost na kratki razdalji, si lahko predstavljamo kot potrebo za vzpostavitev osebnega prostora, na katerem imajo plenilci boljši pogled in nadzor nad plenom. Sočasno pa je prostor namenjen tudi za umik pred udarci plena, saj se ob preveliki natrpanosti ni mogoče umakniti.

Na plenilca vplivajo sila privlačnosti plena, sila odbojnosti ostalih plenilcev na razdalji *da* in sila trenja s tlemi.

Drugi del simulacije je nadgradnja prvega, kjer poskušamo simulirati napad na plen. Poleg sil iz prvega dela vpliva na plenilce še sila vidnega kota v obliki odbojnosti. Najprej po pravilih iz prvega dela ujamemo plen, sledi obkoljevanje z razporejanjem plenilcev in glavni cilj napad na plen, dokler ga ne onesposobimo. Plenilec se poskuša približati znotraj območja branjenja *cd*, če se nahaja izven vidnega kota plena in če je od zadnjega udarca minilo več kot predviden čas *hit_frequency*, ga lahko ponovno udari. V primeru, da se plenilec nahaja znotraj območja branjenja *cd* in je v vidnem kotu α, obstaja možnost, da bo prejel udarec. Ob prejetem udarcu simuliramo poškodbo in s tem poškodovanega plenilca odmaknemo izven območja branjenja *cd* za določen časovni interval. Po pretečenem času, se plenilec lahko vrne v borbo s plenom. Plenilci vnaprej poznajo vidni kot plena, tako se lahko, glede na njegovo usmerjenost, pravočasno vrnejo na varno razdaljo. Primarni cilj naloge je onesposobiti plen, katerega obkoljujemo dokler ne pade. S sekundarnim ciljem pa želimo ugotoviti ali s spreminjanjem parametrov lahko simuliramo alfa par (samca in samice), ki sta dominantna člana

skupine. Ugotoviti želimo ali je skupina plenilcev tako bolj uspešna.



Slika 3.2: Sile, ki vplivajo na plenilca.

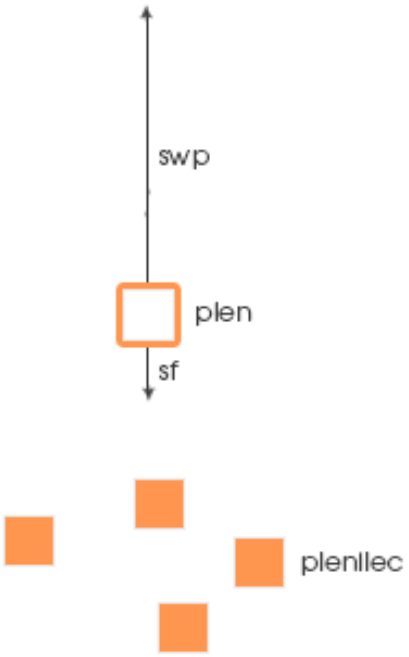
Slika 3.2 prikazuje sile, ki vplivajo na plenilca: sila privlačnosti plena **sp**, sila odbojnosti plenilcev **sw**, sila odbojnosti vidnega kota plena **sc** in sila odbojnosti v obliki trenja tal **sf**.

3.2.2 Opis plena

V prvem delu simulacije je cilj plena glede na priložen vir [1] definiran zelo preprosto. Plen se ne odziva na spremembe v okolju, ne zaznava plenilcev in preprosto stoji na željeni poziciji $\mathbf{u}_i$.

V drugem delu simulacije je osnovni cilj preživetje plena. Glavni cilj je določen glede na trenuten test. Plen ima v vsakem izmed testov definirana območja: območje zaznavanja in območje branjenja, prav tako mu definiramo število življenj HP (angl. *health point*). Plen se zaveda prisotnosti plenilca, ki se nahaja v območju zaznavanja. Kot smo že omenili imajo vsi osebki definirano trenutno lokacijo in smerni vektor gibanja. Na plen vplivajo različne sile: sila pregona v obliki odbojnosti, ki jo ustvarjajo plenilci nad plenom in sila trenja s tlemi. Cilj plena je preživetje, zato skozi ves čas poskuša pobegniti, ko plenilci vstopijo v območje zaznavanja *vd*, se plen prične braniti. Branjenje plena simuliramo tako, da plen konstantno preverja, kateri

plenilec se nahaja najbližje in se obrne proti njemu. Plen ne spreminja smeri dokler se njegov pogled ne poravna s prejšnjo lokacijo plenilca, ki si jo je zapomnil pred rotacijo. Plen ima v naprej določen vidni kot α in odmik kota β, ki predstavlja območje kjer se plenilec prične odmikati. Vidni kot α je območje, ki ga vidi plen in je prav tako območje kamor udari, ko se brani. Cilj plena je odgnati plenilce in preživeti dlje od predpisanega časa.



Slika 3.3: Sile, ki vplivajo na plen.

Na sliki 3.3 lahko vidimo vpliv sile na plen, ki jo ustvarjajo plenilci nad plenom **swp** v obliki odbojnosti in silo trenja s tlemi **sf**.

3.2.3 Opis branjenja plena

Ko plenilci obkolijo plen, je naravni instinkt, da žival poskuša poiskati izhod, če plan pobega ne uspe, se prične braniti. Žival je ponavadi zmedena, zato poskuša storiti vse, da prežene napadalce. V simulaciji simuliramo branjenje po principu odženi najbližjega. Plen se zaveda okoliščin, zato v vsakem trenutku pozna lokacije napadalcev znotraj območja zaznavanja, tako se vedno

poskuša obrniti proti tarči, ki ji je najbližja. Frekvenca rotiranja in hitrost sta časovno in hitrostno omejena, da se izognemo sunkovitemu spreminjanju rotacije, v primeru ko sta dva ali več osebka na isti oddaljenosti od plena. Če se plenilec nahaja znotraj območja branjenja *cd* in če je od zadnje obrambe minilo več kot predviden čas *hit_frequency*, plen simulira napad, ter tako poškoduje plenilca. Plen ima določeno število življenj HP, ob izgubi vseh življenje je žival onesposobljena.

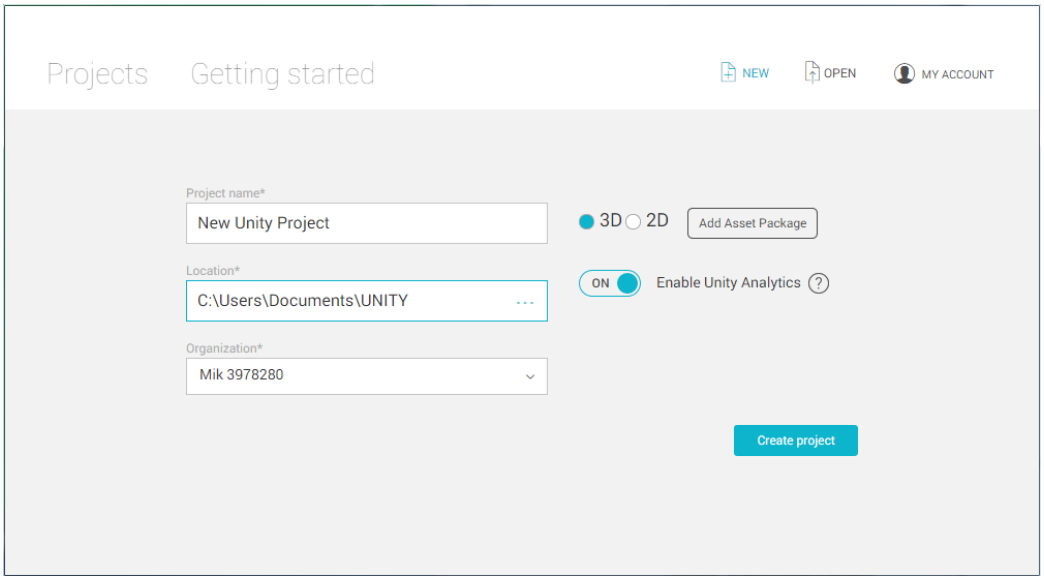
Poglavje 4

Izdelava simulacije

Simulacijo za lažjo nadgradnjo ločimo na več delov, s tem izboljšamo nadzor in pregled nad kodo za kasnejše spreminjanje. Simulacijo razdrobimo na dve glavni skripti `Wolf_behaviour` in `Prey_behaviour`, kjer znotraj prve pišemo kodo za nadzor vseh plenilcev, medtem ko je druga skripta namenjena plenu. Znotraj razredov definiramo spremenljivke, ki jim v funkciji `Start()` nastavimo začetne vrednosti. V `FixedUpdate()` poskrbimo za zbiranje informacij iz okolja, znotraj katere na podlagi pridobljenih informacij izračunamo vektor gibanja in posodobimo premike vseh prisotnih objektov v simulaciji.

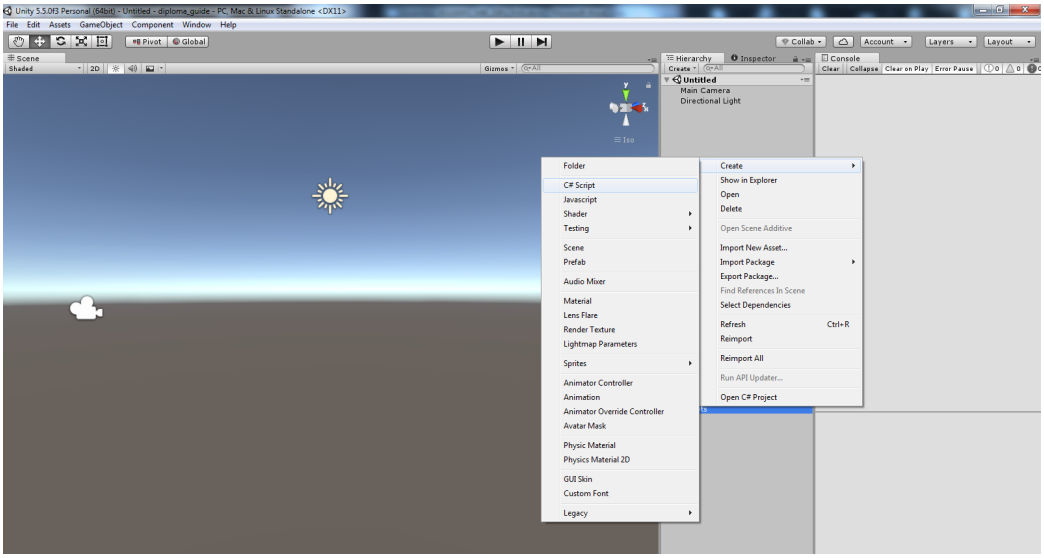
4.1 Ustvarjanje projekta

Ob zagonu programa Unity nam uvodno okno ponudi izbiro med 2D in 3D projektom, ter shranjevanje na disk ali v oblak (angl. *cloud*). Pred kreiranjem projekta lahko naložimo dodatke (angl. *assets*) ali pa jih dodamo kasneje po potrebi.



Slika 4.1: Unity, ustvarjanje projekta.

Ustvarimo projekt z osnovno sceno, ki vsebuje luč in glavno kamero, kot je prikazano na sliki 4.1. V okvir projekt (angl. *project*) na začetku ustvarimo mape v katere kasneje dodajamo skripte, materiale, teksture, 2D ali 3D modele in ostale potrebne elemente za pripravo scene.



Slika 4.2: Unity, ustvarjanje skripte.

Na sliki 4.2 lahko vidimo ustvarjanje prazne C# skripte, katera vsebuje razred in dve osnovni funkciji: `Start()` s katero nastavimo oziroma inicializiramo začetne vrednosti spremenljivk in se izvede samo v prvi sličici (angl. *frame*), medtem ko se funkcija `Update()` izvede vsako sličico. Fiziko in delo s silami računamo v funkciji `FixedUpdate()` zaradi fiksne dolžine časa, ki preteče med sličicami. Funkcija `Update()` ni primerna za izračune, saj pretečen čas med sličicami niha, kar lahko vpliva na izračune, saj vsak računalnik deluje z različno hitrostjo.

```
void Start () {
    print ("Start:" + Time.deltaTime);
}
void Update () {
    print ("Update:"+Time.deltaTime);
}
void FixedUpdate(){
    print ("FixedUpdate:"+Time.deltaTime);
}
```

```
Start: 0.02
UnityEngine.MonoBehaviour:print(Object)
Update:      0.01657082
UnityEngine.MonoBehaviour:print(Object)
Update:      0.01656526
UnityEngine.MonoBehaviour:print(Object)
FixedUpdate: 0.02
UnityEngine.MonoBehaviour:print(Object)
Update:      0.01656654
UnityEngine.MonoBehaviour:print(Object)
FixedUpdate: 0.02
UnityEngine.MonoBehaviour:print(Object)
Update:      0.01656825
UnityEngine.MonoBehaviour:print(Object)
```

Slika 4.3: Konsistentnosti med funkcijami.

Na sliki 4.3 lahko vidimo klic kode, ter izpis konzole iz katere je razvidno, da se funkcija `Start()` požene samo prvič, in je vedno konsistentna. `Update()` se požene za klicem funkcije `Start()`, iz katerega pa lahko takoj vidimo, da čas klica med sličicami niha. `FixedUpdate()` pa je zmeraj konsistentna, zato je primerna za natančne izračune. Znotraj teh treh funkcij pišemo in kličemo vso programsko kodo.

4.2 Ustvarjanje scene

Najprej je potrebno v sceno dodati objekte, ki predstavljajo plen in plenilce. To storimo tako, da za vse prisotne objekte izberemo kar primitivni 3D model v obliki kocke. Ustvarimo dve oznaki (angl. *tag*): `wolf` in `prey`, ter jih pripnemo ustreznemu objektu. Kameri nastavimo pogled iz ptičje perspektivne na os X in Z, ter zaklenemo rotacijo za boljši nadzor nad sceno. Kot smo že omenili na začetku poglavja, v mapi skripte ustvarimo 2 skripti v katere pišemo kodo za manipulacijo plenilcev in plena.

4.3 Prvi del diplomske naloge

V tem podpoglavju razložimo in implementiramo algoritem za obkoljevanje plena. Implementiramo premikanje plena, plenilcev in razložimo vpliv sil na posameznika.

4.3.1 Premikanje plenilcev

Uporabimo preprosti model na osnovi delcev, kjer plenilce $(1, ..., N)$ in plen (p) definiramo kot delec, na katerega vplivajo sile privlačnosti in odbojnosti. Model temelji na osnovi Newtonovega zakona o gibanju, ki ga zapišemo v obliki vektorjev:

$$m_i \dot{\mathbf{v}}_i = \mathbf{F}_i, \qquad i = p, 1, ..., N \tag{4.1}$$

Iz enačbe (4.1) razberemo parametre: m_i predstavlja težo osebka, $\mathbf{v}_i$ pospešek oziroma vektor gibanja, $\mathbf{F}_i$ pa nam pove kakšen je seštevek sil, ki vplivajo na posameznika. Premik posameznika simuliramo tako, da k trenutni poziciji $\mathbf{u}_i$ prištevamo vrednost hitrosti $\mathbf{v}_i$:

$$\dot{\mathbf{u}}_i = \mathbf{v}_i, \qquad i = p, 1, ..., N \tag{4.2}$$

Hitrost $\mathbf{v}_i$ predstavlja seštevek sil, ki vplivajo na plenilca i:

$$\dot{\mathbf{v}}_i = \frac{1}{m_i}(C_W^P \frac{\mathbf{u}_p - \mathbf{u}_i}{\|\mathbf{u}_i - \mathbf{u}_p\|^2} \left(1 - \frac{dc^2}{\|\mathbf{u}_i - \mathbf{u}_p\|^2}\right) - \sum_{j=1, j\neq i}^{N} C_W^W \frac{\mathbf{u}_j - \mathbf{u}_i}{\|\mathbf{u}_i - \mathbf{u}_j\|^2} \phi_{i,j} - C_f\ \mathbf{v}_i), \qquad i = 1, ..., N \tag{4.3}$$

Za boljše razumevanje enačbo (4.3) razdelimo na več delov in jih predstavimo v naslednjih podpoglavjih. Unity teče s časovnim korakom 0.02s zato moramo silo pomnožiti s prirejenim časovnim korakom *custom_timestamp*, da se približamo časovnemu koraku iz vira [1]. Vrednost spremenljivke *custom_timestamp* lahko spreminjamo za pohitritev ali upočasnitev elementov v sceni.

4.3.2 Sila trenja

Najprej na podlagi trenutne vrednosti hitrost $\mathbf{v}_i$ izračunamo silo trenja z enačbo:

$$\mathbf{F}_{friction} = -C_f\ \mathbf{v}_i \tag{4.4}$$

Sila trenja deluje v obliki odbojnosti, saj ima vedno nasprotno smer od smeri gibanja.

4.3.3 Sila plena

Da se začnemo premikati, moramo najprej ustvariti začetno silo, ki nas želi pripeljati iz ene točke v drugo. Izračunamo silo privlačnosti med plenilcem in plenom, ter tako ustvarimo silo gibanja v smeri plena:

$$\mathbf{F}_{i,p} = C_W^P \frac{\mathbf{u}_p - \mathbf{u}_i}{\|\mathbf{u}_i - \mathbf{u}_p\|^2} \left(1 - \frac{dc^2}{\|\mathbf{u}_i - \mathbf{u}_p\|^2}\right) \tag{4.5}$$

Parametri $C_W^P, m_i,\ dc$ so konstantne vrednosti skozi čas, ki jih nastavimo na začetku simulacije. Vektor $\mathbf{u}_p$ in $\mathbf{u}_i$ pa predstavljata trenutno lokacijo

plena in trenutno lokacijo i-tega plenilca. Parameter C_W^P je koeficient privlačnosti, s katerim lahko nastavljamo kako močno nas plen privlači. Enačba (4.5), nam vrne vektor, ki ga prištejemo k trenutnemu seštevku sil.

4.3.4 Seštevek sil plenilcev

$$\mathbf{F}_{i,j} = \sum_{j=1, j\neq i}^{N} C_W^W \frac{\mathbf{u}_j - \mathbf{u}_i}{\|\mathbf{u}_i - \mathbf{u}_j\|^2} \phi_{i,j} \tag{4.6}$$

$\mathbf{F}_{i,j}$ je seštevek vseh sil, ki jih plenilci ustvarjajo nad plenilcem i. Parameter C_W^W je koeficient, ki nam pove kakšna je odbojnost med plenilci. $\phi_{i,j}$ predstavlja Gaussovo funkcijo, ki je odvisna od relativne lokacije plenilca i in plenilca j, ter kritične razdalje *da*. Funkcija nam pove na kakšni razdalji bo odbojnost med plenilcema največja. Vrednost Gaussove funkcije je največja kadar sta oba plenilca oddaljena od plena za vrednost *da* in drastično upade, ko se eden izmed njiju oddalji od vrednosti *da*:

$$\phi_{i,j} = \exp(-cw\ ((Ri - da)^2\ +\ (Rj - da)^2)) \tag{4.7}$$

Enačbo (4.6) smo implementirali tako, da na začetku simulacije v funkciji `Start()` poiščemo vse plenilce brez samega sebe in jih shranimo v seznam wolfs, ki vsebuje GameObjects. Vsak časovni korak se sprehodimo skozi tabelo in pimerjamo trenutnega plenilca i z vsemi plenilci iz tabele, kjer plenilca iz tabele označimo z j. Za vsakega izmed njih izračunamo oddaljenost med j in p (plen), ter jo zapišemo v spremenljivko R_j. Sledi izračun razdalje med trenutnim plenilcem j in p, ter jo zapišemo v spremenljivko R_i. Gaussovi funkciji podamo R_i, R_j, kritično razdaljo *da* in *cw*. Parameter *cw* predstavlja širino Gaussove funkcije, ki vpliva na odbojnost med plenilci.

Funkcija za izračun seštevka vseh sil plenilcev po enačbi (4.6) v programu Unity:

```
public Vector3 Calc_wolfs_force(Vector3 up){
  Vector3 sum = Vector3.zero;
  Ri = Vector3.Distance (up, ui_i);
  for (int i = 0; i < wolfs.Count; i++) {
    float w_i_gauss = Calc_gaussian (
```

```
        c_w,
        Ri,
        Vector3.Distance (up,
          wolfs[i].GetComponent<Wolf_behaviour>().ui_i),
          da
        );
        sum += (c_ww/mi) *
        Calc_g(wolfs[i].GetComponent<Wolf_behaviour>().ui_i, ui_i) * w_i_gauss;
                }
  return sum;
}
```

Funkcija za izračun vrednosti Gaussa po enačbi (4.7):

```
public float Calc_gaussian(float c_w, float Ri, float Rj, float da){
  return Mathf.Exp((-1 * c_w) * (Mathf.Pow((Ri - da), 2) + Mathf.Pow((Rj - da), 2)));
}
```

4.3.5 Premikanje plena

Premikanje plena implementiramo na podoben način. Najprej v funkciji `Start()` poiščemo vse plenilce z oznako wolf in jih dodamo v seznam vseh plenilcev. V vsakem koraku se izvede funkcija `FixedUpdate()` v kateri najprej izračunamo silo trenja tako, kot smo to storili pri plenilcih, nato pa za vsakega izmed plenilcev v tabeli izračunamo silo pregona v obliki odbojnosti, ter jo prištejemo v skupni seštevek sil, od nje odštejemo silo trenja. Tako dobimo pospešek plena $\mathbf{v}_p$ po naslednji enačbi:

$$\dot{\mathbf{v}_p} = \sum_{i=1}^{N} C_P^W \frac{\mathbf{u}_p - \mathbf{u}_i}{\|\mathbf{u}_i - \mathbf{u}_p\|^2} - C_f \mathbf{v}_p \tag{4.8}$$

Hitrost $\mathbf{v}_p$ v vsakem koraku prištevamo trenutni poziciji $\mathbf{u}_i$ in tako poustvarimo premikanje plena. Seštevek sil plenilcev pomnožimo z -1, saj želimo, da se odmikamo v nasprotno smer, kot ga lovijo plenilci.

4.4 Drugi del diplomske naloge

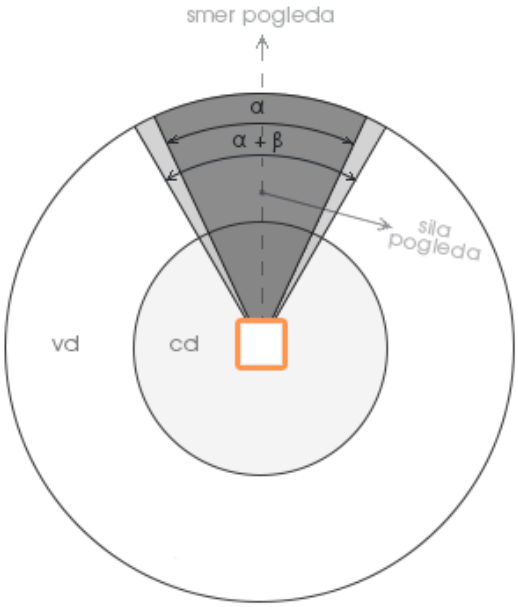
V tem poglavju razložimo in implementiramo nadgradnjo prvega dela diplomske naloge, ki ji dodamo algoritem napada na plen. Implementiramo logiko vida in branjenje plena, ter logiko napada plenilcev na plen.

4.4.1 Sila vidnega kota plena

Za drugi del diplomske naloge se enačba hitrosti $\mathbf{v}_i$ za posameznega plenilca nekoliko spremeni. Nadgradnja enačbe $\mathbf{v}_i$:

$$\dot{\mathbf{v}}_i = C_W^P \frac{\mathbf{u}_p - \mathbf{u}_i}{\|\mathbf{u}_i - \mathbf{u}_p\|^2}\left(1 - \frac{dc^2}{\|\mathbf{u}_i - \mathbf{u}_p\|^2}\right) - \sum_{j=1,j\neq i}^{N} C_W^W \frac{\mathbf{u}_j - \mathbf{u}_i}{\|\mathbf{u}_i - \mathbf{u}_j\|^2}\phi_{i,j} + \left(C_C^P \ \mathbf{pdir}\left(1.5 - \frac{pca}{\frac{\alpha}{2} + \beta}\right)\right) - C_f \mathbf{v}_i, i = 1, ..., N \tag{4.9}$$

Vidni kot je sestavljen iz kota α, odmika kota β in vd, ki predstavlja dolžino kamor sega pogled plena. Sila vidnega kota ima vpliv na plenilca, kadar se nahaja znotraj vidnega območja.



Slika 4.4: Vidni kot plena in prikaz sile pogleda, ki vpliva na plenilca.

Za izračun sile vidnega kota moramo najprej izračunati točko v smeri pogleda plenilca **popf**. To storimo s klicem funkcije `transform.forward`, ki nam vrne smerni vektor **pfow**. Točko **popf** dobimo tako, da seštejemo

vektor trenutne lokacije in smerni vektor **pfow**, ki je pomnožen z razdaljo R_i. Kot smo že omenili R_i predstavlja trenutno razdaljo med plenom in plenilcem i.

Nato izračunamo smerni vektor sile **pdir**, tako da od trenutne lokacije plenilca i odštejemo **popf**, ter dobljeno vrednost normaliziramo za lažje nadaljnje računanje.

Sledi izračun trenutnega kota *pca* med pogledom plena in trenutne lokacije plenilca. Zato poskrbi funkcija `Calc_angle`, ki sprejme 3 parametre v obliki pozicij: lokacija plena, točka v smeri pogleda plena **popf** in lokacija plenilca. Na podlagi teh treh točk izračunamo trenutni kot *pca* po kosinusnem izreku. S spodnjo enačbo izračunamo faktor vpliva na silo vidnega kota, ki nam pove kako močno nas sila potiska vstran od centra vidnega kota glede na našo trenutno pozicijo:

$$cone_{fac} = 1.5 - \frac{pca}{\frac{\alpha}{2} + \beta} \tag{4.10}$$

Silo vidnega kota izračunamo po enačbi (4.11), kjer vektor smeri **pdir** pomnožimo s faktorjem vpliva $cone_{fac}$ in željenim koeficientom vidnega kota C_C^P , ki ga nastavimo sami.

$$\mathbf{F}_{cone} = \left(C_C^P \ \mathbf{pdir} \ \left(1.5 - \frac{pca}{\frac{\alpha}{2} + \beta} \right) \right) \tag{4.11}$$

4.4.2 Napad na plen

Plenilec se približuje plenu in neprestano preverja ali je znotraj območja zaznavanja plena *vd*. Ob vstopu v območje *vd* prične s preverjanjem ali ga plen vidi ali ne. To storimo tako, da ob vsakem koraku pokličemo funkcijo `Calc_angle`, če je vrnjen kot manjši od polovice vidnega kota plena, pomeni, da nas plen vidi. Plenu se lahko približujemo kljub temu, da nas vidi, če smo zunaj območja branjenja *cd*. V primeru, da se nahajamo zno-

traj območja branjenja in nas plen vidi, se odmikamo, dokler nismo zunaj kritičnega območja, hkrati pa se poskušamo izmakniti vidnemu kotu, tako da se odmikamo vstran od središča pogleda plena. Kadar se nahajamo znotraj vidnega polja plena, spremenimo vrednost varnostne razdalje *dc* na *dc_safe*, ki ga izračunamo po enačbi:

$$dc\_safe = cd + (0.8\ m_p); \tag{4.12}$$

Vrednost *dc_safe* izračunamo na podlagi dolžine polmera območja branjenja plena *cd*, kateremu prištejemo težo plena pomnoženo z vrednostjo 0,8. Predpostavljamo, da težje živali zadajo močnejše udarce, zato na podlagi teže in območja zaznavanja *vd* določimo varnostno razdaljo *dc*.

Kadar se nahajamo znotraj območja branjenja *cd* in vidnega kota plena, obstaja možnost, da nas je plen udaril, in če nas je, simuliramo poškodbo. Stanje spremenljivke *set_hit* spremenimo na resnično. Ko je stanje *set_hit* resnično, nastavimo čas kazni *time_penalty*, ki jo izračunamo tako, da pomnožimo koeficient kazni udarca *time_penalty_coeficient* in število prejetih udarcev doslej *n_hit*. Dokler ima plenilec kazen, je vrednost spremenljivke *dc* nastavljena na *dc_penalty*, ki predstavlja prepoved približevanja plenu. Izračunamo jo po enačbi:

$$dc\_penalty = cd + 0.3\ n\_hit\ m_p\ \left(\frac{time\_penalty\_left}{time\_penalty}\right) \tag{4.13}$$

Plenilec lahko plen udari, ko se nahaja znotraj kritične razdalje *da* pod pogojem, da ga plen ne vidi. Če plenilcu uspe udariti plen, se plenilcu nastavi zakasnitev udarca. Plenilec lahko ponovno udari plen, ko se zakasnitev izniči.

4.4.3 Branjenje plena

V spremenljivko *dt* shranjujemo pretečen čas od začetka simulacije. Spremenljivko *dt* v vsakem koraku posodobimo za vrednost časovnega koraka

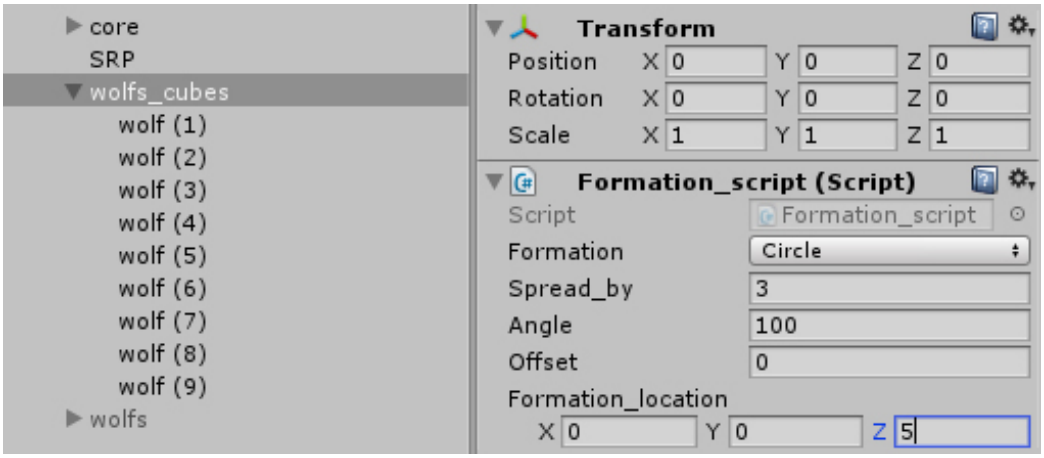
0.02s in preverimo ali smo v zadnjem času izvedli rotacijo. To storimo tako, da od *dt* odštejemo čas zadnje pojavitve rotacije. Vrnjen čas nam pove, koliko časa je preteklo od zadnje rotacije. V primeru, da je pretečen čas večji od frekvence rotacije *fr*, pričnemo z iskanjem plenilcev. Najprej preverimo koliko plenilcev se nahaja znotraj območja zaznavanja *vd* in jih vpišemo v seznam. Za vsakega izmed kandidatov v seznamu izračunamo razdaljo med njima. Najbližjega plenilca nastavimo za tarčo in simuliramo rotiranje. To storimo tako, da funkciji `Quaternion.RotateTowards` posredujemo 3 parametre: trenutno rotacijo plena, trenutno lokacijo tarče in hitrost rotacije s katero želimo rotirati plen. Plen se obrne proti zadnji lokaciji, kjer se je nahajal plenilec. Ob vsakem koraku preverjamo ali je plenilec še v vidnem kotu α s klicem funkcije `Calc_angle`. V primeru, da je vrnjen kot manjši od polovice vidnega kota plenilca, pomeni da je plenilec znotraj kritičnega območje in vidnega kota plena. Če v zadnjih nekaj sekundah nismo udarili, potem izvršimo udarec. Plenilcu nastavimo stanje spremenljivke *set_hit* na resnično in ga začasno izključimo iz napada. Postopek ponovimo, ko lahko izvedemo naslednjo rotacijo.

4.5 Urejevalnik postavitev

Med izdelavo simulacije je potrebno veliko sprotnega testiranja, zato je bilo potrebno plenilce in plen neprestano dodajati, odvzemati, spreminjati atribute, prestavljati na različne položaje itd., kar je zelo zamudno in nepraktično. Za boljšo produktivnost sprogramiramo gradnik, ki bo ponujal različne možnosti postavitev, dodajanje ali odvzemanje, aktiviranje ali deaktiviranje posameznih skript in nastavljanje različnih postavitev plenilcev.

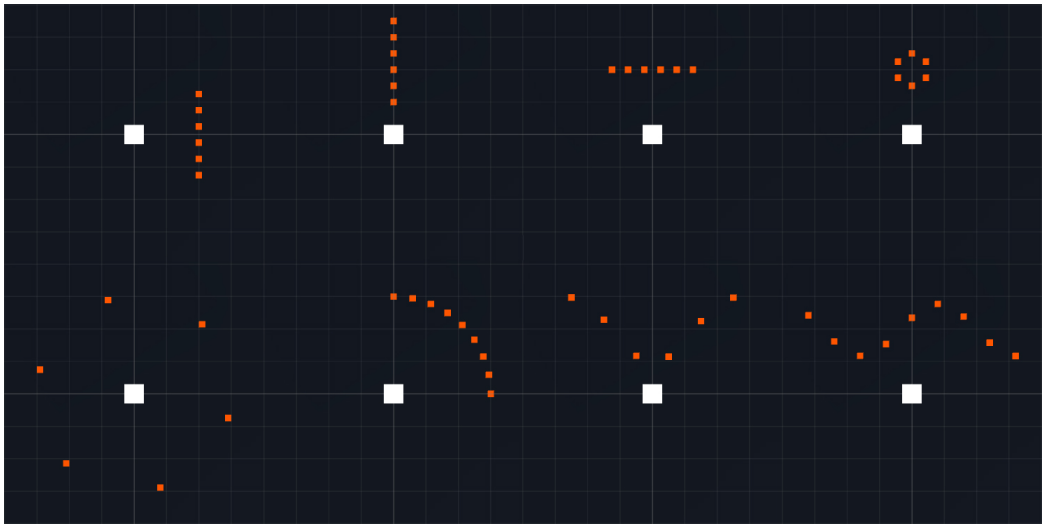
Najprej naredimo nov razred *formation_script*, ki vsebuje javne spremenljivke (glej sliko 4.5). Ustvarimo razširjen razred *formation_editor* iz nadrejenega razreda Editor. Skripte izpeljane iz tega razreda lahko izvajajo kodo brez zagona projekta, ki se izvajajo znotraj scene. Za ta namen uporablja funkcijo `OnSceneGUI( )`, ki nam omogoči manipulacijo nad sceno.

Za izdelavo različnih formacij, uporabimo označbo wolf, ki smo jo ustvarili na začetku projekta in pripeli vsem primernim objektom. V razredu *formation_editor* smo dodali selektor, ki nam v seznam zapiše vse objekte, ki vsebujejo podano označbo. Nad elementi iz seznama izvajamo različne matematične operacije glede na izbrani način postavitve iz menija, ki smo ga pred tem definirali v razredu *formation_script*. Na formacijo vplivajo različni parametri, ki jih lahko spreminjamo v realnem času. Tako smo definirali horizontalno, vertikalno, krožno, vijugasto, ter vsestransko postavitev *paper_equations*, ki pokriva vse teste iz vira [1] (glej sliko 4.6). Vsaka izmed možnosti vsebuje *formation_location*, s katerim nastavimo izhodiščno točko, *spread_by* nam pove kakšen naj bo razmik med elementi, angle predstavlja kot, offset pa zamik kota.



Slika 4.5: Urejevalni vmesnik, za manipulacijo nad plenilci.

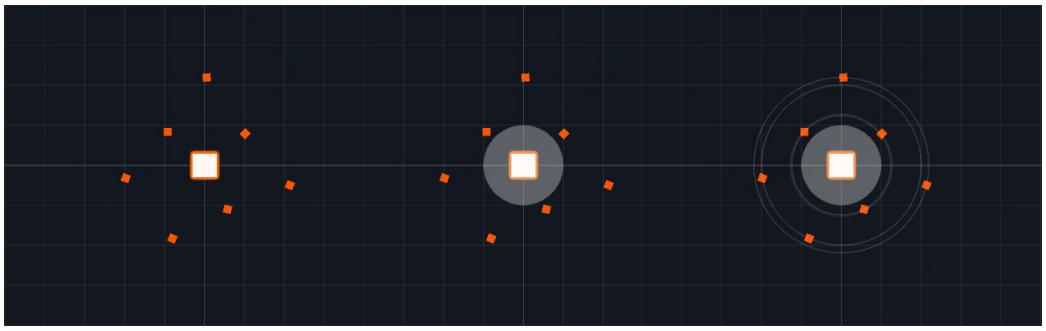
Prisotnim plenilcem v sceni nadredimo starša, ki je prazni objekt (angl. *empty object*), na katerega pripnemo urejevalno skripto, kar nam omogoči dostop do urejevalnega vmesnika.



Slika 4.6: Rezultati različnih postavitev s pomočjo urejevalnika.

4.6 Vizualizacija

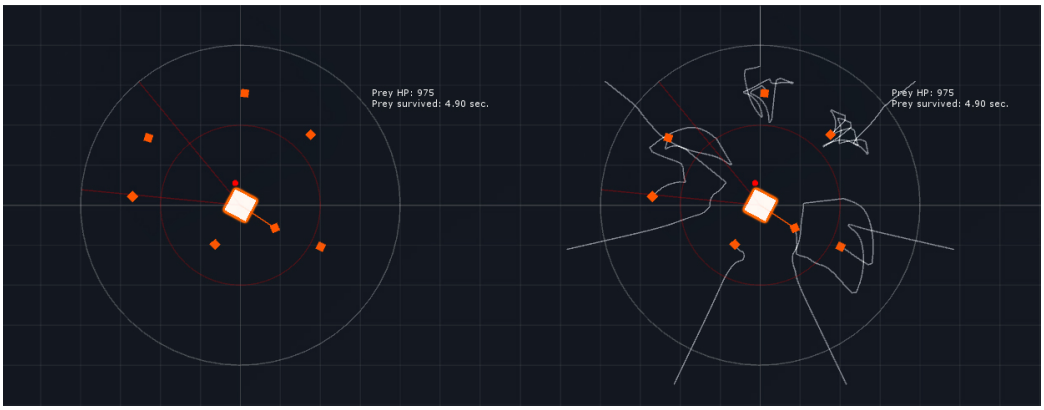
Vizualizacija je zelo pomemben del simulacije, ki nam nudi hiter in enostaven pogled nad izvajanjem trenutnega testa. Za potrebe primerjave rezultatov smo najprej uporabili zelo preprost pristop, kjer oranžne kocke predstavljajo plenilce, plen pa je bel.



Slika 4.7: Različni prikazi vizualizacije za boljšo preglednost nad rezultati.

Slika 4.7 prikazuje različne vizualizacije, ki nam pomagajo pri preglednosti simulacije. Skrajno levo lahko vidimo golo simulacijo, medtem ko srednji

prikaz prikazuje kritično območje *dc*, skrajno desni pa nadgradnjo srednje, kjer dodamo razdalje R_i v obliki orbit.



Slika 4.8: Vizualizaciji simulacije lova.

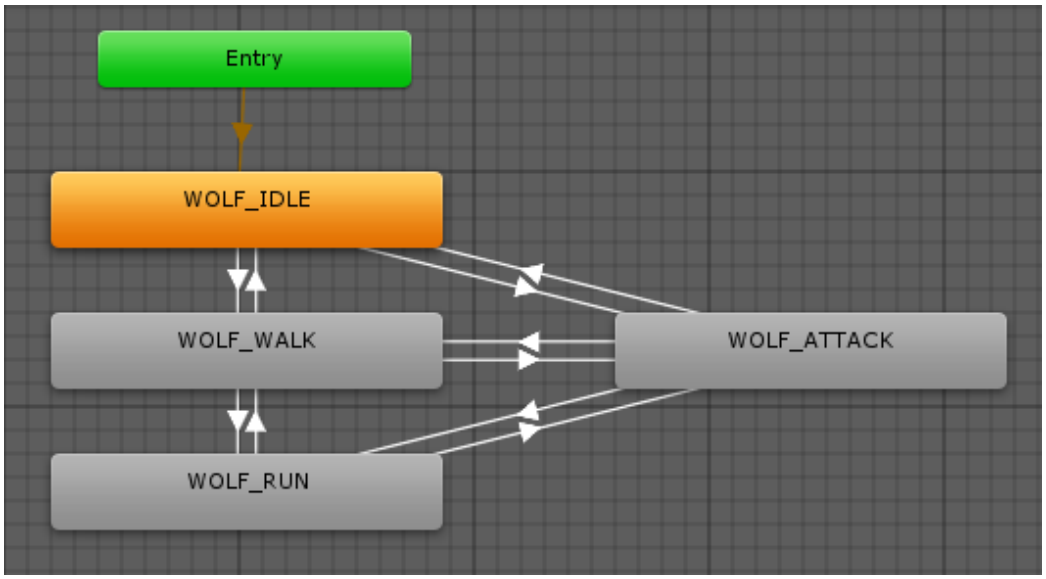
Na levi strani slike 4.8 vidimo simulacijo lova, kjer notranji krog predstavlja območje branjenja *cd*, zunanji krog pa predstavlja območje zaznavanja *vd*. Trikotnik rdeče barve predstavlja vidno polje plena. Povezava oranžne barve pa simulira udarec plenilca. Kroglica rdeče barve predstavlja središče SRP, ki je skupno središče vseh plenilcev. Zgoraj desno ob plenu prikažemo stiatistiko lova, kjer prey HP predstavlja število življenj plena, prey survived pa predstavlja čas obleganja plenilcev. Na desni strani slike 4.8 pa lahko vidimo nadgradnjo, kjer dodamo sled bele barve, ki označuje pot posameznega plenilca, tako lahko enostavno spremljamo kje se gibajo plenilci. Za boljšo razločnost sledem nastavimo material, ki nam omogoči nastavitev barve za posamezno sled.

4.6.1 Nadgradnja simulacije v 3D

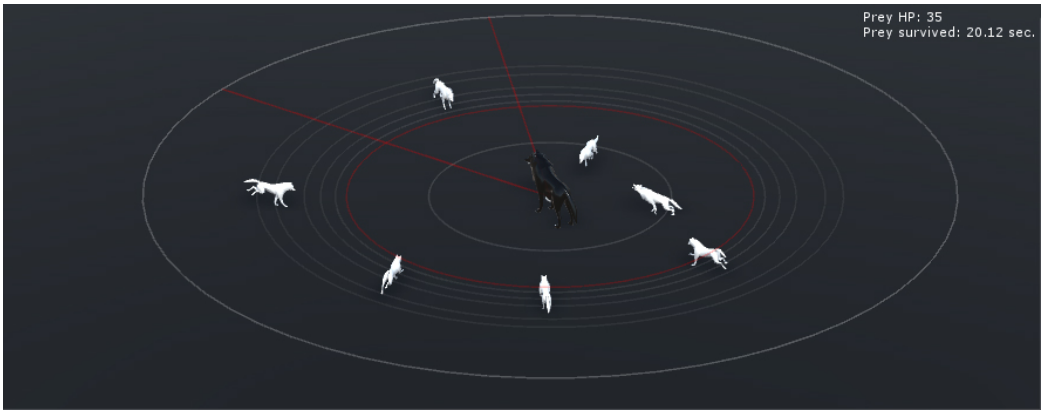
Za zbiranje podatkov in opravljanje testov, je najbolj preprosti pristop najboljši, saj je cilj simulacije v tej fazi predvsem natančnost rezultatov posameznega testa, zato smo za poustvarjanje rezultatov iz vira [1] uporabili čim bolj poenostavljene primitivne oblike objektov.

Končen izgled simulacije pa nadgradimo s 3D objekti, ki vsebujejo skelet

na podlagi katerega je ustvarjena animacija s programom Maya. 3D objekte uvozimo v Unity in ga zamenjamo z obstoječim izgledom kocke. Za delovanje animacije je potrebno dodati komponento animator, ki nam omogoči animiranje objekta. V mrežo komponente za animiranje dodamo željene animacije, ki jih želimo uporabiti v simulaciji. Za naše potrebe potrebujemo 4 stanja: stanje mirovanja, hoje, teka in napada. Diagram odločanja si lahko ogledamo na sliki 4.9. Vsaka animacija ima začetno vstopno stanje (angl. *Entry state*), ki skrbi za prehod na prevzeto stanje animacije. V našem primeru nastavimo mirovanje (angl. *idle*) za privzeto stanje. To je stanje v katerega se vračamo vedno, kadar ne izpolnjujemo pogojev za prehod na ostale animacije. Diagram odločanja poteka na podlagi dveh parametrov: hitrost plenilca in stanja napada, ki jih nastavimo v skripti `Wolf_behaviour`. Če je hitrost plenilca večja od 0.05 potem preidemo v stanje hoje (angl. *walk*), medtem ko za stanje teka potrebujemo hitrost večjo od 0.4. Iz vsakega stanja, lahko preidemo v stanje napada, če je vrednost spremenljivke za napad resnična, vračanje iz stanja napada pa je odvisno od trenutne hitrosti premikanja po napadu.



Slika 4.9: Urejevalnik animacij, `animator`.



Slika 4.10: Končna simulacija plenjenja z animacijo in 3D modelom volka.

Poglavje 5

Rezultati in testi

Najprej smo uporabili predlagane enačbe iz vira [1], s katerimi smo uspešno poustvarili gibanje plenilcev v smeri plena. Na minimalni varnostni razdalji *dc*, bi se morali plenilci odmikati drug od drugega in se tako razvrstiti okrog plena. Na razdalji *dc*, se je zgodilo ravno obratno, plenilci so se povlekli v gručo in se pričeli premikati kaotično. S spremembo parametra *cw* na negativno vrednost in Gaussove funkcije, smo uspeli poustvariti obkoljevanje. Ob prednastavitvi parametrov iz vira [1, tabela 1] in vrednosti $dc = 1$, bi morali poustvariti rezultate, ki nam vrnejo strukturo obkoljevanja, kjer se plenilci postavijo v več krogov. S spremembo Gaussove formule nam je uspelo poustvariti postavitev, kjer se plenilci postavijo v en krog. Ne glede na vnešene parametre nam ne uspe poustvariti rezultatov brez, da bi spremenili staro enačbo:

$$\mathbf{F}_i = \mathbf{F}_{i,p} + \sum_{j=1, j\neq i}^{N} \mathbf{F}_{i,j} - C_f \, \mathbf{v}_i \tag{5.1}$$

5.1 Rešitev napačne formule

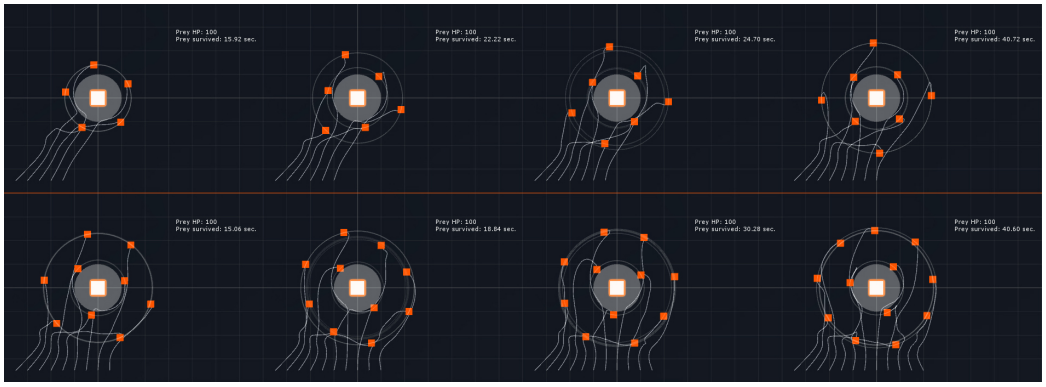
Najprej smo pričeli razvijati simulacijo na podlagi priloženih enačb iz vira [1]. Ob zagonu simulacije opazimo, da so se plenilci pričeli premikati proti plenu. Glede na vir [1], bi se plenilci morali na razdalji *da* odmikati od ostalih in

sprožiti akcijo obkoljevanja okrog plena. Na podlagi priloženih enačb, smo uspeli plenilce pripeljati do razdalje *da*, ob kontaktu z minimalno varnostno razdaljo *dc*, so se plenilci pričeli obnašati ravno obratno, kar je sprožilo tesno formiranje v gručo, sledilo pa je kaotično obnašanje. Ob pregledu enačb in priloženih parametrov iz vira [1, tabele 1], lahko hitro opazimo, da je enačba napačna, saj je avtor predpostavil, da so vsi koeficienti pozitivne vrednosti. Napaka je v viru [1], natančneje v enačbi (3.3), kjer za posameznega plenilca izračunamo Gaussovo funkcijo, ki nam vrača negativno vrednost. Ugotovimo, da se ob spremembi predznaka koeficienta *cw*, plenilci pričnejo na razdalji *da* odmikati drug od drugega in obkolijo plen. Vendar to ne odpravi težave za $N > 5$, kjer N predstavlja število plenilcev. Za vrednost $dc = 1$ in $N = 6$, bi se morali plenilci razporediti v dva obroča okrog tarče, kar z enačbo iz vira [1] ni bilo mogoče doseči. Ob kontaktu avtorja, smo prišli do skupnega spoznanja, da so v viru [1] napačne enačbe, in sicer (3.4) in (3.5). Posledično (3.3) za plenilce in (3.8), ter (3.7) za plen, saj funkcije ne vračajo pravilnih vrednosti glede na priložene rezultate, zato jih z originalnimi enačbami ni mogoče poustvariti.

V pričakovanju na avtorjev odgovor, smo že sami prišli do zelo podobne rešitve. Avtor se je zelo hitro odzval z naslednjo explicitno enačbo, ki združuje enačbi (3.2) in (3.3), ter zamenja enačbi (3.4) in (3.5) iz vira [1] z novimi. Nova enačba nas pripelje do boljših rezultatov:

$$\dot{\mathbf{v}}_i = \frac{1}{m_i}(C_W^P \frac{\mathbf{u}_p - \mathbf{u}_i}{\|\mathbf{u}_i - \mathbf{u}_p\|^2}\left(1 - \frac{dc^2}{\|\mathbf{u}_i - \mathbf{u}_p\|^2}\right) - \sum_{j=1, j\neq i}^{N} C_W^W \frac{\mathbf{u}_j - \mathbf{u}_i}{\|\mathbf{u}_i - \mathbf{u}_j\|^2}\phi_{i,j} - C_f\ \mathbf{v}_i), \qquad i = 1, ..., N \tag{5.2}$$

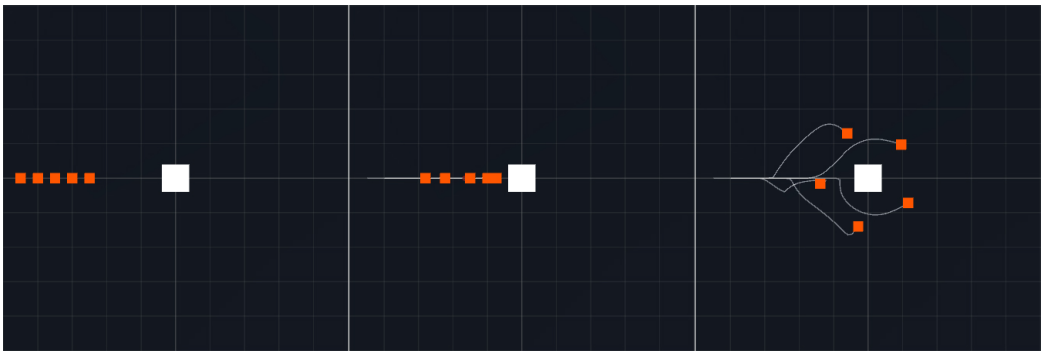
Glede na predlagano enačbo za pospešek plenilca, smo izpeljali enačbo za pospešek plena. S pomočjo dveh novih formul smo prišli do drugačnih vendar še zmeraj zelo podobnih rezultatov.



Slika 5.1: Poustvarjanje simulacije obkoljevanja z novo enačbo.

Za naše potrebe drugega dela diplomske naloge, je bilo potrebno ustvariti več nivojsko obkoljevanje, kar smo s pomočjo novo izpeljanih enačb uspešno in brez problemov poustvarili. Zgolj iz radovednosti smo želeli ugotoviti, zakaj prihaja do razhajanja pri rezultatih iz priloženega vira [1, slika 5], kjer poskušamo poustvariti periodično rotacijo plenilcev. Pod vplivom specifičnih nastavitev parametrov skozi čas naraste nestabilnost med plenilci, katera sproži proces, ki izgleda kot napetost med plenilci in povzroči rotacijo. Ker z enakimi vrednostmi rezultatov nismo uspeli poustvariti, smo najprej poskušali s spreminjanjem parametrov, nato smo predvidevali, da napetost ne naraste dovolj zaradi premajhne velikosti decimalnih mest zapisa lokacij. Unity za svoje namene ne potrebuje dvojne natančnosti (angl. *double*), saj za računanje kolizij med objekti in računanje z vektorji zadostuje enojna natančnost. Najprej je bilo potrebno napisati vse funkcije za računanje vektorjev z dvojno natančnostjo, nato smo ponovno poskušali s testiranjem. Z izvajanjem testov ugotovimo, da je dolžina zapisa enojne natančnosti dovolj natančna. Posvetujemo se z avtorjem, ki nam predlaga, da je velika možnost problem uporabe različnega okolja, saj je avtor svoje delo ustvaril v programskem jeziku Fortran [8] operacijskega sistema Linux [9] in simulacijo izrisoval s knjižnjico pgplot [10]. Zaradi sintakse jezika Fortran [8], ki nam je popolnoma neznana se naše raziskovanje prvega dela diplomske naloge tukaj zaključi.

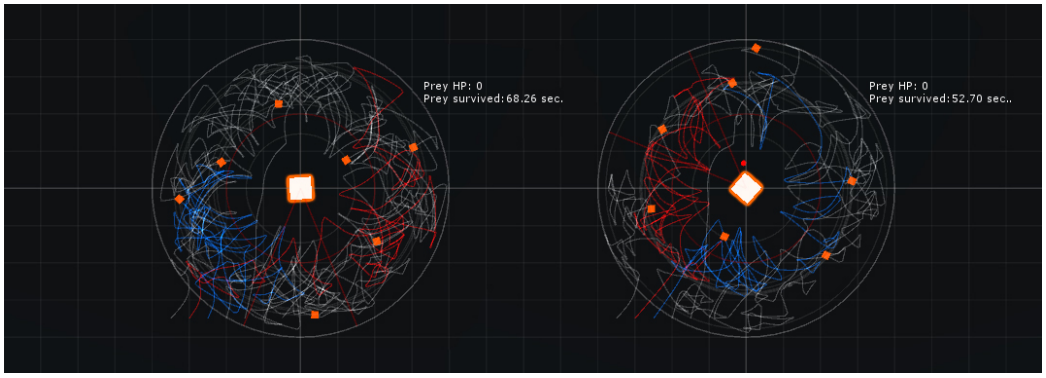
5.2 Problem poravnave



Slika 5.2: Prikaz problema in rešitve vodoravne postavitve.

Avtor vira [1] izpostavi problem mrtve točke, kadar poravnamo plenilce in plen v vodoravno ali navpično linijo, kot prikazuje slika 5.2 na skrajno levi strani. Srednji del prikazuje problem mrtve točke, kjer se plenilci na minimalni varnosti razdalji dc ne ločijo drug od drugega, ter posledično ne sprožijo akcije obkoljevanja pravilno. Problem nastane, kadar je seštevek odbojne sile, ki jo izvajajo plenilci nad plenilcem i enak nič za koordinato Z ob vodoravni postavitvi, ter za koordinato X ob navpični postavitvi. Oddaljevanje med plenilci se zgodi, vendar samo v smeri postavitve, za ničte vrednosti seštevka odbojne sile pa faktor Gaussa nima vpliva, saj se pomnoži z nič. Problem enostavno odpravimo z pregledovanjem seštevka sil nad plenilcem i, če je sila enaka 0 v določenem primeru, potem vrednost te spremenljivke nastavimo na minimalno pozitivno število enojne natančnosti, ki preverjeno deluje za vrednosti do 10^{-43} pri časovnem koraku *custom_timestamp* vrednosti 0,01. Rezultat rešitve je samostojno odločanje najbližjega plenilca, ko pride do takšne situacije. Plenilce se premakne v smeri obkoljevanja za zelo majhno vrednost, kar sproži akcijo obkoljevanja. Prikaz delujoče simulacije s pomočjo implementirane rešitve lahko vidimo na skrajno desni strani slike 5.2.

5.3 Primerjava napada na plen z in brez alfa para



Slika 5.3: Primerjava enakovredne skupine levo, ter skupina z alfa parom na desni.

Med preučevanjem nove formule za obkoljevanje plena smo ugotovili, da lahko s prametrom *cw* ustvarimo hierarhijo skupine plenilcev. To storimo tako, da posameznemu plenilcu nastavimo specifično vrednost parametra *cw*. Če se parameter *cw* dovolj razlikuje med plenilci lahko s tem ustvarimo hierarhijo znotraj skupine plenilcev, ter tako ustvarimo alfa par.

Zanima nas ali je skupina z ali brez alfa para bolj uspešna?

Test izvedemo tako, da plenilcem nastavimo začetne vrednosti iz tabele 5.1, plenu pa glede na vrednosti iz tabele 5.2. Prva skupina plenilcev vsebuje enakovredne člane, zato vsem nastavimo vrednost $cw = 0,3$. Druga skupina vsebuje dva plenilca, ki ju poimenujemo alfa1 in alfa2, ki imata vrednost $cw = 1$, ostalim plenilcem pa nastavimo *cw* na 0.3. V obeh primerih imajo plenilci začetno pozicijo definirano z (0.5 * i - 4, 0, -3.5), kjer i predstavlja zaporedno številko plenilca, vrednosti pa predstavljajo X,Y in Z koordinate, medtem ko plen postavimo na izhodišče koordinatnega sistema (0, 0, 0). Plenilci so definirani kot nepremagljivi, plen pa ima 100 enot energije. Posamezen

custom_timestamp	časovni korak	0.01
m_i	teža plenilca	0.1
C_f	koeficient trenja za plenilce	1
C_W^P	koeficient sile, ki jo ustvarja plen nad plenilcem	1
C_W^W	koeficient sile, odbojnost med plenilci	0.5
d_c	začetna varnostna razdalja plenilca	1
d_a	razdalja do plena, na kateri se pričnejo plenilci odmikati drug od drugega	1.5
cone_tolerance_deg	odmik vidnega kota plena β v stopinhaj	20
cw	širina koeficienta Gaussove funkcije øij (1/sqrt(2 * cw))	0.5
N	število plenilcev v testu simulacije	3-12
time_penalty_ coefficient	čas kazni plenilca ob udarcu plena	5

Tabela 5.1: Začetni podatki plenilca.

udarec plenilca odstrani 5 enot energije. Test izvedemo tako, da plenilci napadajo plen toliko časa, dokler ne izgubi vseh energije. Test izvedemo za $N = 3, 5, 7$ in 11, kjer N predstavlja število plenilcev.

Najprej smo izvedli test po zgornjih pravilih za skupino brez alfa para, kot lahko vidimo na sliki 5.5 primer a), nato smo izvedli test skupine z alfa parom, kot prikazuje primer b), ter dobili zanimive rezultate.

Iz rezultatov na sliki 5.4 je razvidno, da imajo plenilci zelo veliko prednost, kadar skupina vsebuje alfa par, saj se na prvi pogled čas lova zelo zmanjša. Alfa par prevzame vlogo glavnih napadalcev, kot je razvidno iz odebeljenih vrednosti stolpca ugrizi, prav tako pa prejemata največ udarcev. Iz tega lahko sklepamo, da sta zelo agresivna in se kljub velikemu številu udarcev zelo hitro vrneta nazaj v boj. Stolpec ugrizi prikazuje število ugrizov posameznega plenilca, poškodbe pa predstavljajo število prejetih udarcev, ki jih

m_p	teža plena	1
C_f	koeficient trenja za plen	2
C_P^W	koeficient sile, ki jo ustvarja plenilec nad plenom	0.2
C_W^{PC}	koeficient sile, ki jo ustvarja plen z vidnim poljem nad plenilcem	0.8
C_{ac}	množitelj koeficienta C_W^{PC}	5
P_{va}	vidni kot plena α	45
R_f	pogostost rotacije plena	1
R_s	hitrost rotacije plena	200
vd	območje zaznavanja	4
cd	območje branjenja	2
HP	število življenj plena	100
$cone\_td$	toleranca vidnega polja v stopinjah	20

Tabela 5.2: Začetni podatki za plen.

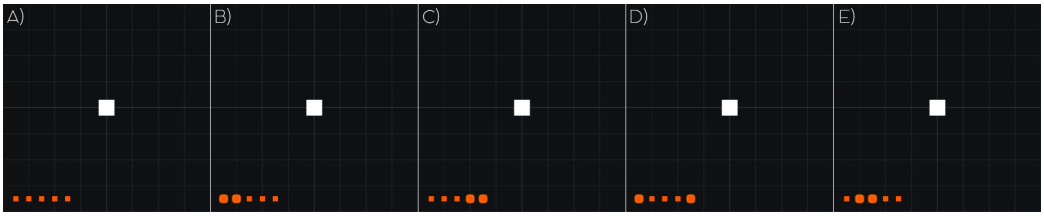
je plenilec prejel od plena, odebeljene vrednost pa predstavljajo vrednosti alfa1 in alfa2. Vrednosti stolpca postavitve sta: equal, ki predstavlja enakovredne plenilce, medtem ko alfa-L prestavlja alfa par lociran na levi začetni poziciji, ter ostale plenilce.

N	postavitev	cw	čas	ugrizi	poškodbe	skupno poškodb
3	equal	0.3	42.28	7, 7, 6	2, 1, 2	5
	alfa-L	1, 0.3	34.34	**7, 5**, 8	**2, 2**, 0	4
5	equal	0.3	48.98	2, 4, 4, 5, 5	0, 1, 1, 0, 1	3
	alfa-L	1, 0.3	35.84	**8, 5**, 2, 3, 2	**2, 2**, 1, 1, 0	6
7	equal	0.3	68.26	2, 5, 2, 1, 4, 2, 4	1, 0, 0, 0, 1, 1, 2	5
	alfa-L	1, 0.3	57.38	**4, 9**, 1, 3, 2, 0, 1	**4, 2**, 1, 0, 1, 0, 0	8
11	equal	0.3	73.17	2, 2, 2, 3, 3, 1, 2, 1, 0, 3, 1	0, 0, 1, 2, 1, 1, 1, 1, 0, 1, 1	9
	alfa-L	1, 0.3	40.30	**5, 5**, 0, 0, 1, 0, 4, 2, 2, 1, 0	**2, 1**, 0, 0, 0, 0, 0, 1, 0, 1, 0	5

Slika 5.4: Rezultati primerjave skupine brez in z alfa parom.

Ker so rezultati zelo specifični, smo želeli preveriti konsistentnost alfa para, če jima spreminjamo začetni položaj, kot lahko vidimo na sliki 5.5, kjer

primer a) predstavlja enakovredne plenilce. V tabeli to prikažemo z označbo equal saj ne vsebujejo alfa para, b) označimo z alfa-L, kjer alfa par lociramo na levo stran skupine, c) postavitev z označbo alfa-R, par lociramo desno, d) z alfa-LR, kjer postavimo plenilca alfa1 skrajno levo, alfa2 pa skrajno desno. Postavitev e) označimo z alfa-C, ki predstavlja postavitev para v sredino, če je število plenilcev liho zavzema alfa2 središče, medtem ko je alfa1 lociran kot levi sosed (glej sliko 5.5).



Slika 5.5: Postavitve začetnih pozicij testiranja.

Teste smo ponovili za pozicije (c-e) in jih primerjali s prejšnjimi rezultati. Ugotovili smo, da alfa par brez dvoma izboljša uspešnost lova. V povprečju se čas lova zmanjša za kar 16 do 31% v odvisnosti od velikosti skupine plenilcev, posledično pa sta zaradi tega večkrat poškodovana. Če primerjamo poškodbe obeh skupin opazimo, da so poškodbe skupine brez alfa para zelo naključne, medtem ko v skupini z alfa parom prevladujejo poškodbe predvsem pri njiju. Ob spremljanju simulacije opazimo, da koeficient *cw* vpliva tudi na minimalno razdaljo, saj se alfa par približa bližje za udarec, kot ostali plenilci. V diplomski nalogi predpostavljamo, da imajo vsi plenilci enako močan ugriz, neglede na moč ali razdaljo udarca. Zelo zanimivo bi bilo preizkusiti kakšne rezultate bi dobili, če bi plenilec za bližji pristop udarca zadal večje poškodbe plenu.

N	postavitev	cw	čas	ugrizi	poškodbe	skupno poškodb
	equal	0.3	42.28	7, 7, 6	2, 1, 2	5
	alfa-L	1, 0.3	34.34	**7, 5**, 8	**2, 2**, 0	4
3	alfa-R	1, 0.3	24.80	7, **7, 6**	0, **0, 1**	1
	alfa-LR	1, 0.3	47.40	**6**, 5, **9**	**3**, 2, **1**	6
	alfa-C	1, 0.3	34.34	**7, 5**, 8	**2, 2**, 0	4
	equal	0.3	48.98	2, 4, 4, 5, 5	0, 1, 1, 0, 1	3
	alfa-L	1, 0.3	35.84	**8, 5**, 2, 3, 2	**2, 2**, 1, 1, 0	6
5	alfa-R	1, 0.3	27.06	1, 3, 4, **6, 6**	0, 2, 0, **1, 1**	4
	alfa-LR	1, 0.3	33.00	**4**, 3, 2, 3, **8**	**3**, 1, 0, 0, **0**	4
	alfa-C	1, 0.3	39.02	3, **5, 4**, 3, 5	1, **2, 2**, 0, 0	5
	equal	0.3	68.26	2, 5, 2, 1, 4, 2, 4	1, 0, 0, 0, 1, 1, 2	5
	alfa-L	1, 0.3	57.38	**4, 9**, 1, 3, 2, 0, 1	**4, 2**, 1, 0, 1, 0, 0	8
7	alfa-R	1, 0.3	40.08	0, 0, 1, 4, 0, **8, 7**	0, 0, 1, 0, 0, **1, 3**	5
	alfa-LR	1, 0.3	45.32	**6**, 1, 1, 1, 4, 1, **6**	**3**, 0, 0, 0, 0, 0, **3**	6
	alfa-C	1, 0.3	52.70	2, 0, **6, 9**, 2, 0, 1	2, 0, **3, 2**, 0, 0, 0	7
	equal	0.3	73.17	2, 2, 2, 3, 3, 1, 2, 1, 0, 3, 1	0, 0, 1, 2, 1, 1, 1, 1, 0, 1, 1	9
	alfa-L	1, 0.3	40.30	**5, 5**, 0, 0, 1, 0, 4, 2, 2, 1, 0	**2, 1**, 0, 0, 0, 0, 0, 1, 0, 1, 0	5
11	alfa-R	1, 0.3	57.52	1, 0, 2, 2, 1, 0, 3, 1, 0, **4, 6**	1, 0, 1, 0, 0, 0, 1, 0, 0, **4, 3**	10
	alfa-LR	1, 0.3	52.48	**6**, 0, 0, 0, 4, 0, 0, 1, 0, 0, **9**	**4**, 1, 0, 0, 1, 0, 0, 0, 0, 0, **2**	8
	alfa-C	1, 0.3	58.46	0, 1, 0, 0, **8, 9**, 1, 0, 1, 0, 0	0, 0, 0, 0, **3, 3**, 0, 0, 0, 0, 0	6

Slika 5.6: Rezultati z alfa parom na različnih pozicijah.

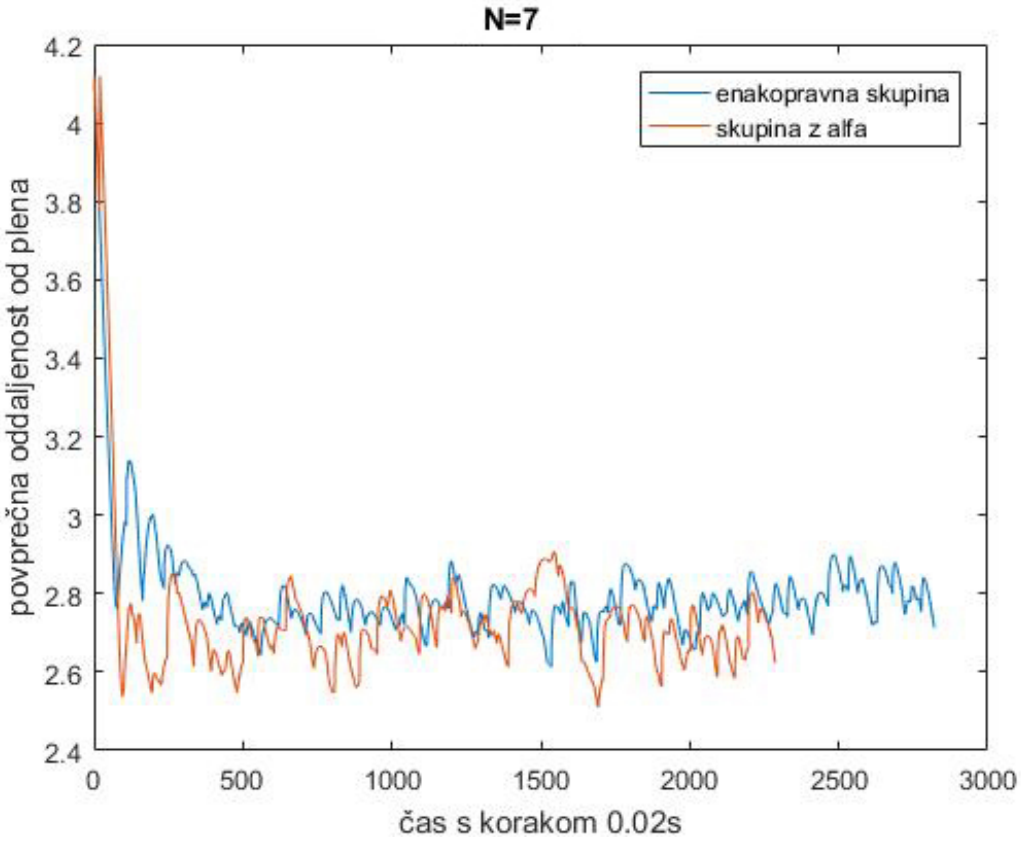
5.3.1 Primerjava grafov obeh skupin

Naslednja stvar, ki jo želimo preveriti je kako agresivna postane skupina kot celota in kako vpliva alfa par na posameznega soseda v skupini.

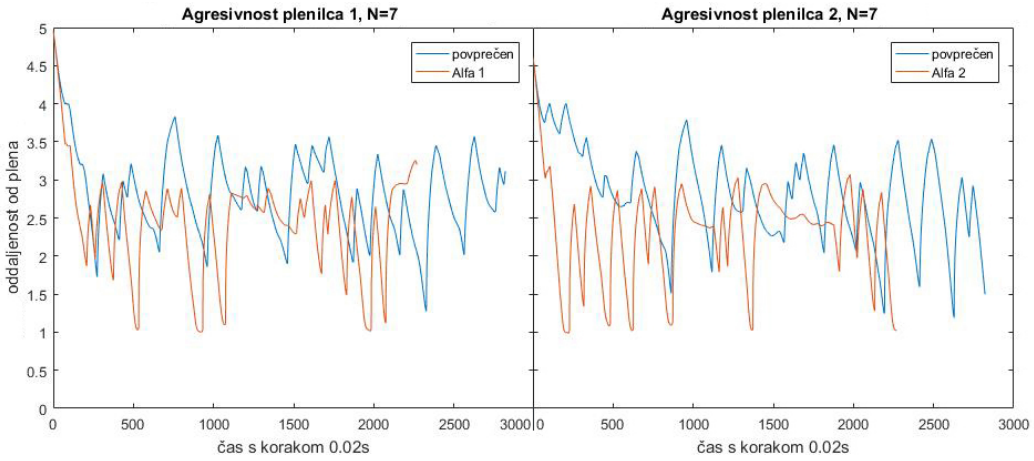
Simulacijo je potrebno nadgraditi tako, da si v vsakem časovnem koraku v datoteke zapisujemo trenutno oddaljenost plenilca od plena v odvisnosti od časa. Zapisujemo si trenutno oddaljenost plenilca alfa1, alfa2, enega izmed sosedov alfa para, plenilca, ki je najdlje od alfa para, ter povprečje oddaljenosti skupine, ki jo izračunamo tako, da seštejemo vse trenutne oddaljenosti od plena, ter jih delimo s številom plenilcev. Pridobljene rezultate vstavimo v program matlab in izrišemo grafe.

Iz grafa slike 5.7 je zelo lepo razvidno, da se skupina z alfa parom približa plenu bližje za kar 6.4%, kar se odraža s predčasno zaključenim bojem. Graf slike 5.8 pa nam zgolj potrdi prejšnje rezultate testiranj, saj se ob spremembi vrednosti *cw* poveča agresivnost plenilca alfa1 za 15.62% in alfa2 za 21.88%. Zaradi njune agresivnosti stojita veliko bližje kot ostali plenilci in sta zato največkrat izpostavljena kritičnemu območju, hkrati pa s tem preusmerjata

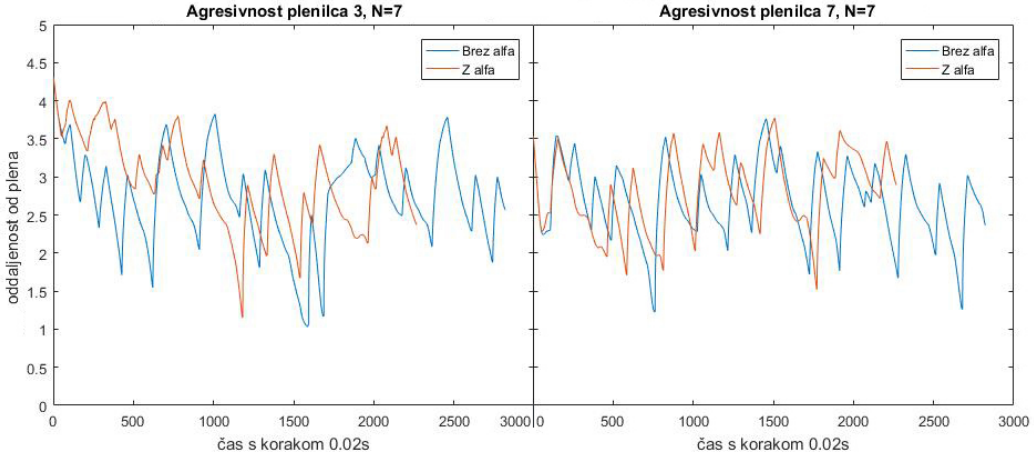
pozornost plena v njuno smer, tako imajo ostali plenilci več možnosti za priložnost ugriza, čeprav so manj izkušeni. Ob izvajanju testov spremljamo simulacijo in opažamo, da so sosednji plenilci alfa para nekoliko zadržani, iz slike 5.9 razberemo upad agresivnosti sosednjih plenilcev za 15.82%, med tem ko plenilcu 7 upade agresivnost za 13.42%, ki se nahaja na drugi strani skupine. Iz tega lahko sklepamo, da so plenilci ob alfa paru manj izkušeni in lahko predpostavimo, da iščejo zaščito ob bolj atleltskem plenilcu, saj se počutijo bolj varno, kar se odraža na njihovi agresivnosti in varnostni razdalji, medtem ko bolj samozavestni plenilci pomagajo obkoljevati plen z druge strani.



Slika 5.7: Graf agresivnosti skupine brez in z dominantnim parom.



Slika 5.8: Primerjava alfa plenilca 1 in 2.

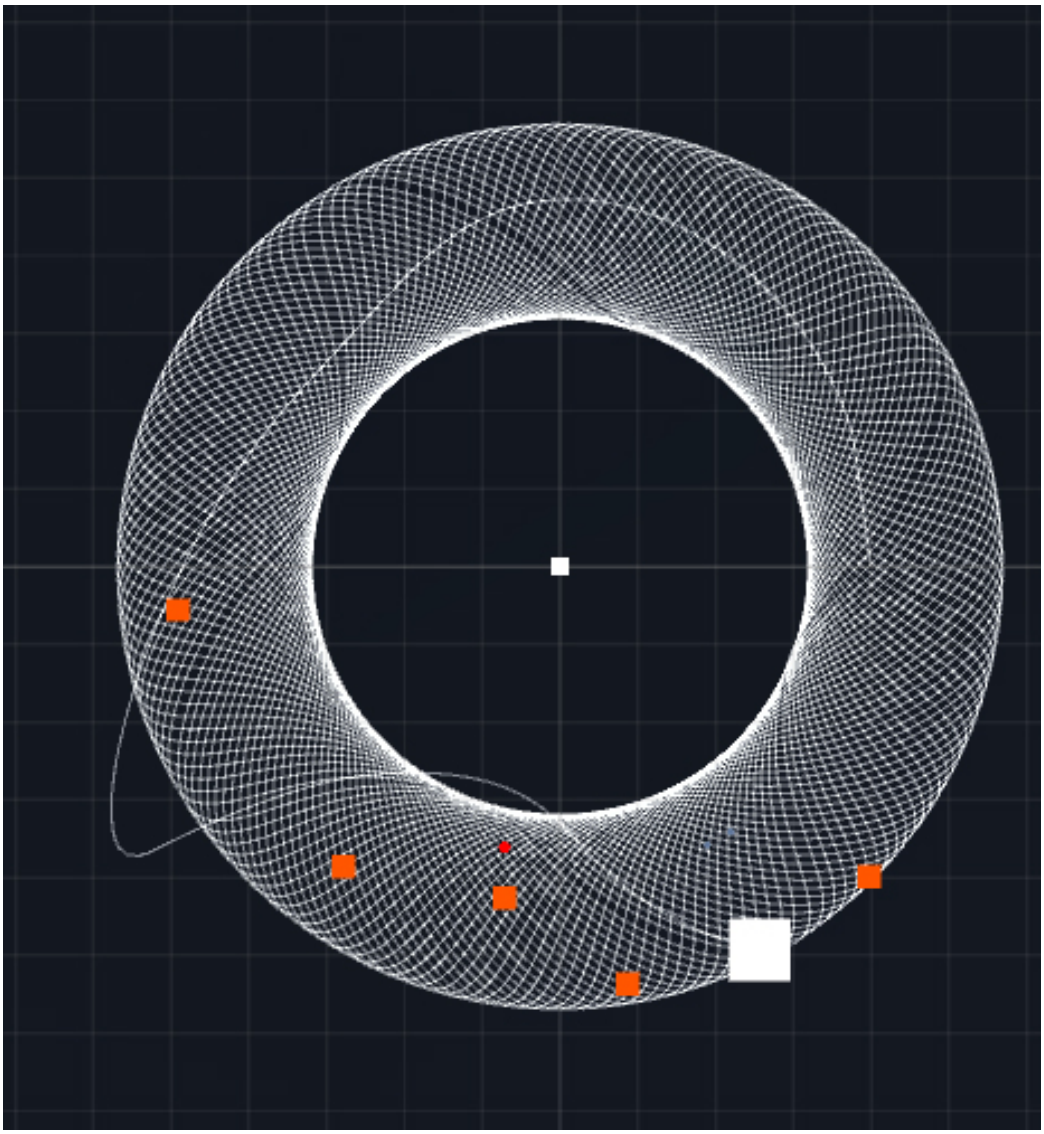


Slika 5.9: Primerjava plenilcev pod vplivom alfa para.

5.4 Zanimivosti

Zgolj iz radovednosti nas je zanimalo ali lahko algoritem popravimo tako, da se plen brani s krožnim gibanjem in s tem odganja plenilce. Simulirati želimo plen, ki brani mladiča v centru koordinatnega sistema. Parametre smo nastavili tako, da se plen giba v krogu oddaljen od centra koordinatnega sistem na razdalji $R_i = 3$, algoritem smo obrnili tako, da plenilci plen privlačijo. Pot

plena je označena z belo črto, rezultat krožnega branjenja pa lahko vidimo na sliki 5.10.



Slika 5.10: Obrnjen algoritem.

Poglavje 6

Sklepne ugotovitve

Po priloženih enačbah iz vira [1] smo najprej implementirali algoritem, ki se je izkazal za nedelujočega. Rezultatov najprej zaradi napačnih enačb in napak pri zapisu testov ni bilo mogoče poustvariti. Za nadaljnje delo je najprej bilo potrebno raziskati vzrok, zakaj se plenilci ne razvrščajo pravilno. V pričakovanju na odgovor avtorja vira [1], smo sami izpeljali enačbo, ki je bila zelo podobna tej, ki nam jo je kasneje posredoval avtor sam. Na podlagi avtorjeve enačbe smo implementirali algoritem s katerim smo uspešno poustvarili obkoljevanje plena in prišli do zelo podobnih rezultatov. Zaradi spremembe formul je prišlo do odstopanj pri rezultatih, kar je razumljivo in tudi razvidno pri obkoljevanju. V nadaljevanju smo pregledali algoritem in ugotovili, da ne deluje za popolnoma poravnane razvrstitve (vodoravne ali navpične) plenilcev in plena. Za takšne primere smo nadgradili algoritem tako, da se zna pravilno odločati tudi v takšnih primerih. Drugi del simulacije smo nadgradili z algoritmom za napadanje plena in primerjali skupino plenilcev z in brez deominantnega para. S spremembo parametrov smo prišli do spoznanja, da lahko zelo preprosto dosežemo hierarhijo skupine plenilcev in s tem simuliramo dominantni par. Rezultati testiranja so pokazali, da imajo skupine z dominantnima parom večjo uspešnost pri lovu, kot pri tistih skupinah, kjer so plenilci enakovredni. Iz rezultatov smo tudi razbrali, da se manj izkušeni plenilci zadržujejo ob dominantnem paru, kar se odraža na

njihovi povečani varnostni razdalji in imajo znižano agresivnost. Za konec simulacijo animiramo s 3D elementi in opazimo, da je dovolj prepričljiva, da bi jo lako eventuelno uporabili v računalniških igrah.

V naši simulaciji smo predpostavili, da imajo vsi plenilci enako moč ugriza. Zelo zanimivo bi bilo preučiti, kakšne rezultate bi dobili, če bi se plenilec učil iz napak, hkrati pa bi bila moč ugriza določena glede na relativno razdaljo od plena.

Literatura

[1] R. Escobedo, C. Muro, L. Spector, and R. P. Coppinger, "Group size, individual role differentiation and effectiveness of cooperation in a homogeneous group of hunters," *Journal of The Royal Society Interface*, vol. 11, no. 95, 2014.

[2] "Unity store." Dosegljivo: https://store.unity.com. [Dostopano: 1. 2. 2017].

[3] S. Lebreton, "Odprtokodno orodje za razhroščevanje Visual Studio Unity tools." Dosegljivo: https://marketplace.visualstudio.com. [Dostopano: 1. 2. 2017].

[4] "Photoshop." Dosegljivo: https://creative.adobe.com. [Dostopano: 1. 2. 2017].

[5] "Matlab." Dosegljivo: https://www.mathworks.com/products/. [Dostopano: 1. 2. 2017].

[6] "Autodesk maya." Dosegljivo: https://creative.adobe.com/products. [Dostopano: 1. 2. 2017].

[7] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH computer graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[8] "Fortran." Dosegljivo: https://gcc.gnu.org/fortran/. [Dostopano: 1. 2. 2017].

[9] "Linux." Dosegljivo: https://en.wikipedia.org/wiki/Linux. [Dostopano: 1. 2. 2017].

[10] "Pgplot graphics subroutine library." Dosegljivo: http://www.astro.caltech.edu. [Dostopano: 1. 2. 2017].

[11] C. Muro, R. Escobedo, L. Spector, and R. Coppinger, "Wolf-pack (canis lupus) hunting strategies emerge from simple rules in computational simulations," *Science Direct*, vol. 88, no. 3, pp. 192–197, 2011.

[12] D. Shiffman, *The nature of code*. The Nature of Code, 2012.

[13] L. D. Mech, D. W. Smith, and D. MacNulty, *Wolves on the Hunt: The Behavior of Wolves Hunting Wild Prey*. University Of Chicago Press, 2015.

[14] D. Haupt, "Tf3dm 3d model." Dosegljivo: http://tf3dm.com. [Dostopano: 1. 2. 2017].

[15] "Law of cosine." Dosegljivo: https://en.wikipedia.org/wiki/Law_of_cosine. [Dostopano: 1. 2. 2017].